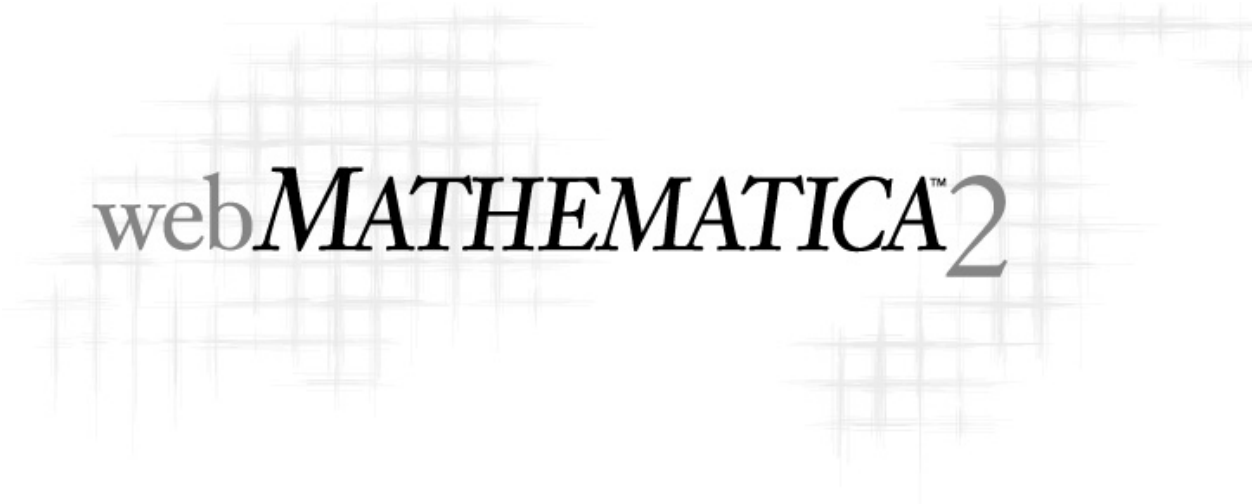


web*Mathematica*:
A User Guide

Tom Wickham-Jones
Version 2.3
January 2006

Copyright © 1999-2006 Wolfram Research, Inc. All rights reserved.

Mathematica, *MathLink*, and *MathSource* are registered trademarks of Wolfram Research, Inc. web*Mathematica*, *The Mathematical Explorer*, *J/Link*, and *MathReader* are trademarks of Wolfram Research, Inc.

Java is a trademark of Sun Microsystems, Inc.

All other trademarks are the property of their respective owners.

Table of Contents

1. Introduction

1.1 What is web*Mathematica?*
1.2 Why Use *Mathematica* in a Web Site?
1.2.1 Computation
1.2.2 Interactive Programming Language
1.2.3 Connectivity
1.2.4 The *Mathematica* Front End
1.2.5 Mathematical Typesetting and MathML
1.3 Why a Web Interface?
1.3.1 Ease of Use
1.3.2 Server-Based Configuration
1.3.3 Web Technologies
1.4 Areas of Use for web*Mathematica*
1.4.1 Web Computation
1.4.2 Education
1.4.3 Publishing
1.4.4 Research
1.4.5 Hobbyist
1.5 web*Mathematica* Technology
1.5.1 Requirements
1.6 web*Mathematica* 2.3 New Features
1.7 web*Mathematica* 2.2 New Features
Support for *Mathematica* 5.1
Database Connectivity
Client Web Services
Authoring System
1.8 web*Mathematica* 2.1 New Features
1.9 web*Mathematica* 2.0 New Features
Support for *Mathematica* 4.2
Simplified Installation

©1988-2006 Wolfram Research, Inc. All rights reserved.

Extended Documentation and Examples
New Templating Mechanism Based on JSP Custom Tags
MathML, SVG, and XML Support
Support for Catching Message and Print Output
Support for HTTP File Upload
Support for HTTP Session Variables
HTML Formatting Functions
1.9.1 Modifications in web*Mathematica* 2.0
Location of Security Configuration File
Location of MSP.conf

2. Installation
2.1 Setting up a Servlet Container
2.1.1 Setting up Java
Unix
Windows
Mac OS X
2.1.2 Setting up Tomcat
Unix
Windows
Mac OS X
2.2 Installing and Configuring *Mathematica*
2.3 Installing the web*Mathematica* Web Application
2.3.1 If You Use Tomcat
2.3.2 If You Do Not Use Tomcat
JRun
Other Servlet Engines
2.4 Configuring for the X Window System (Unix only)
2.4.1 Configuring Xvnc and web*Mathematica*
Install Xvnc
Launch Xvnc
Test Xvnc
Configure web*Mathematica*
2.4.2 Other X Related Issues
Connecting to the X Server
Xvfb
Manual Font Installation
2.5 Upgrading from web*Mathematica* 2.2
2.5.1 Install *Mathematica*
2.5.2 Install the web*Mathematica* Web Application
2.5.3 Configure the New Layout
web.xml
MSP.conf
Security Configuration
2.5.4 Move Content to the New Layout
2.5.5 Finalize the Installation

©1988-2006 Wolfram Research, Inc. All rights reserved.

2.6 Optional Further Configuring
2.6.1 MSP *Mathematica* Application
2.6.2 Launching web*Mathematica* Automatically
Unix
Windows
2.6.3 Web Server Connections
Apache and Tomcat
Microsoft Servers and Tomcat
Apache/Microsoft Servers and JRun
2.7 Testing

3. Basic Examples
3.1 Hello.jsp
3.2 Working with Variables: Variables.jsp
3.3 MSP Functions: Expand.jsp
3.4 Graphics: Plot.jsp
3.5 Typeset Images: Integrate.jsp
3.6 Live 3D Plotting: Plot3DLive.jsp
3.7 Getting Messages: Messages.jsp
3.8 Returning General Content: Content.jsp
3.9 Applets: TextApplet.jsp
3.10 JavaScript: PlotScript.jsp
3.11 Setting Variables: SetBasic.jsp
3.12 Getting Variables: GetBasic.jsp
3.13 Interacting with MSPs: IncludeMSP.jsp

4. Developing Your Own Pages
4.1 Tips and Tricks
4.1.1 Variables
4.1.2 Coding in Pages
4.2 Templates
4.2.1 Browse Examples
4.2.2 Design Examples
4.2.3 Banners and Buttons
4.3 Minimal Installation
4.3.1 Minimal File Layout

5. Applications
5.1 XML
5.1.1 Introduction to XML
XML Compliance
5.1.2 *Mathematica* Support for XML
5.1.3 web*Mathematica* XML Applications
5.2 MathML
5.2.1 Embedding MathML in Web Documents
XHTML
XHTML and MathML
Rendering XHTML and MathML Documents

©1988-2006 Wolfram Research, Inc. All rights reserved.

5.2.2 Generating MathML from web*Mathematica*
MathML Integrate Example
5.2.3 Sending MathML to web*Mathematica*
MathML Input Example
5.3 SVG
5.3.1 Plotting with SVG
5.3.2 SVG Animations
5.4 HTML Formatting
5.4.1 The HTML Functions
HTMLTableForm
HTMLFormat
HTMLSelect
HTMLCheckBox
5.4.2 web*Mathematica* Examples
Table Formatting
Select Formatting
5.5 Using Java APIs
5.5.1 Server APIs
5.5.2 Other Java APIs
5.6 Data Loading and Computation
5.6.1 File I/O
5.6.2 HTTP Upload
5.6.3 Database Connection
5.6.4 Web Service
5.6.5 Data Examples
Loading Data: Load.jsp
Uploading Data: Upload.jsp
Session Storage of Data: Session.jsp
Database Connections: Database.jsp
5.7 *Mathematica* Packages and Applications
5.7.1 Loading Packages
5.7.2 Writing Packages
5.7.3 Installing Packages
web*Mathematica* Applications
$BaseDirectory
$UserBaseDirectory
The Script Directory
$TopDirectory
PackagesDirectory
Absolute Filename
5.8 Alternative Server Technologies
5.8.1 JavaServer Pages
5.8.2 PHP

©1988-2006 Wolfram Research, Inc. All rights reserved.

5.9 PDF Documents
Generating a *Mathematica* Notebook
Converting to PostScript
Converting PostScript to PDF
5.9.1 Creating PDF Example
5.10 Returning General Content
5.10.1 MSPReturn
5.10.2 MSPURLStore
5.10.3 MSPPageOptions/ContentType

6. Advanced Topics
6.1 Variables
6.1.1 Input Variables
Interpretation of Input Variables
Interpreted versus Noninterpreted Values
MSPBlock versus MSPToExpression
6.1.2 Page Variables
6.1.3 Session Variables
6.2 Security
6.2.1 MSP Function Validation
Important: Security Warning
The Validation Process
6.2.2 Setting Your Own Security Model
6.2.3 ToExpression Validation
6.2.4 Access Restrictions
6.3 Evaluation Formatting
Automatic Formatting
MSPFormat
String Formatting
Graphics and Image Formatting
Suppressing Output
Multiple Calculations
6.4 Processing a JSP
6.5 Multiple Kernel Pools
6.5.1 Mapping JSPs to Pools
6.5.2 Multiple Web Applications
6.6 Mapping URLs onto JSPs
6.7 *Mathematica* Initialization
6.8 Logging
6.9 The Kernel Monitor
6.10 Debugging web*Mathematica*
6.11 Displaying Mathematics and Graphics
MSP Functions Returning Images
6.12 Including Static Files

©1988-2006 Wolfram Research, Inc. All rights reserved.

7. Troubleshooting

7.1 Initial Checks
- 7.1.1 Check the Server
- 7.1.2 Check the URL
- 7.1.3 Check the Initial Page
- 7.1.4 Check the Kernel Monitor
- 7.1.5 Check the Log File
- 7.1.6 Check the Console Shell
- 7.1.7 Check *Mathematica*

7.2 Specific Problems
- 7.2.1 Problems Running the Front End
- 7.2.2 Problems Testing Xvnc (Unix only)
- 7.2.3 Problems Testing Xvfb (Unix only)
- 7.2.4 Images Do Not Work
- 7.2.5 *Mathematica* Packages and Applications
- 7.2.6 Kernel Initialization
- 7.2.7 Vertical Alignment in Formatting
- 7.2.8 Timeout Problems
- 7.2.9 UnsatisfiedLinkError
- 7.2.10 Cannot Load JLink`
- 7.2.11 NoClassDefFoundError: TryCatchFinally
- 7.2.12 NoClassDefFoundError: JLink Classes
- 7.2.13 NoSuchMethodError: KernelData
- 7.2.14 Cannot Find jawt.dll (Windows)

7.3 Reporting Problems

8. Classic web*Mathematica* Technology

8.1 Converting MSP Scripts to JSPs
- 8.1.1 Converting an MSP to a JSP
- 8.1.2 Converting a JSP to an MSP

8.2 Multiple Kernel Pools and MSP Scripts

8.3 Processing an MSP Script
- 8.3.1 HTTP and MSP Scripts
- 8.3.2 The MSP Servlet
  - Determine the Script Name
  - Get a *Mathematica* Kernel
  - Assign Input Variables
  - Process Page and Mathlets
  - Postprocess
  - Return the Response
  - Process Images
- 8.3.3 Overview

8.4 Mapping URLs onto MSP Scripts
- 8.4.1 Script Directories

8.5 MSP Scripts and Static Files

©1988-2006 Wolfram Research, Inc. All rights reserved.

8.6 Example MSP Scripts
8.6.1 A Beginning: Hello.msp
8.6.2 Working with Variables: Variables.msp
8.6.3 Using MSP Functions: Expand.msp
8.6.4 Graphics: Plot.msp
8.6.5 Typeset Images: Integrate.msp
8.6.6 Live 3D Plotting: Plot3DLive.msp
8.6.7 Returning General Content: Content.msp
8.6.8 Applets: TextApplet.msp
8.6.9 Calling Back to Java: Request.msp
8.6.10 Uploading a File: UploadFile.msp

9. Appendix
9.1 *Mathematica* Function Reference
9.1.1 **MSPBlock**
Exceptions
9.1.2 **MSPException**
9.1.3 **MSPExportImage**
9.1.4 **MSPFormat**
HTML Formatting
Image Formatting
MathML
9.1.5 **MSPGetMessages**
9.1.6 **MSPGetPrintOutput**
9.1.7 **MSPGetUploadFile**
Exceptions
9.1.8 **MSPGetUploadFileList**
Exceptions
9.1.9 **MSPInclude**
Exceptions
9.1.10 **MSPLive3D**
9.1.11 **MSPPageDirectory**
9.1.12 **MSPPageOptions**
ContentType
MinimumVersion
9.1.13 **MSPReturn**
9.1.14 **MSPRootDirectory**
9.1.15 **MSPSetDefault**
Exceptions
9.1.16 **MSPSessionVariable**
9.1.17 **MSPShow**
9.1.18 **MSPShowAnimation**
9.1.19 **MSPToExpression**
Exceptions
9.1.20 **MSPURLStore**

©1988-2006 Wolfram Research, Inc. All rights reserved.

9.1.21 MSPValue
Exceptions
9.1.22 MSPValueQ
9.1.23 $ExportImageOptions
9.1.24 $MSPFormatType
9.1.25 $PageWidth
9.1.26 $PathInfo
9.1.27 $QueryString
9.1.28 $ScriptName
9.1.29 $ServletConfig
9.1.30 $ServletRequest
9.1.31 $ServletResponse
9.1.32 $ShowAnimationOptions
9.1.33 $TypesetImageWidth
9.1.34 $WebApplication
9.2 MSP Taglib Reference
9.2.1 msp:allocateKernel (tag open)
Determine the Pool
Allocate the Kernel
Assign Input Variables
Assign Parameters
Initialization
9.2.2 msp:evaluate
9.2.3 msp:allocateKernel (tag close)
Java Exceptions
MSPReturn
Set ContentType
Clean the Kernel
Release the Kernel
9.2.4 msp:set
9.2.5 msp:get
9.2.6 msp:includeClassicMSP
9.2.7 msp:forwardClassicMSP
9.2.8 Overview
9.3 Site Configuration
9.3.1 MSP.conf
CheckToExpression
CollectStreams
FileCheckPeriod
FrontEndLaunchFlags
FrontEndLaunchName
ImageAltText
JLinkNativeLibraryDirectory
KernelAcquireLimit
KernelConnectLimit

©1988-2006 Wolfram Research, Inc. All rights reserved.

KernelInitialization
KernelNumber
KernelPools
KernelTimeLimit
MathLinkArguments
MSPDirectory
PackagesDirectory
PoolPath
RequestLog
SecurityConfigurationFile
StaticFileExtensions
VerboseLogs
WebApplicationName
9.3.2 Security Configuration
9.3.3 X Server Configuration
9.4 LiveGraphics3D
9.5 Dynamic HTML
9.5.1 Server Technology
CGI Scripting
Active Server Pages
Server Plug-ins
Perl Scripting
Java Servlets and JavaServer Pages
9.5.2 Client Technology
HTML
JavaScript
Applets
Future Developments
9.6 MathML Technology
9.6.1 MathPlayer
9.6.2 WebEQ
9.6.3 Mozilla
9.6.4 Amaya
9.7 File Layout
web*Mathematica* Web Application
MSP *Mathematica* Application
9.8 Links
9.8.1 *Mathematica* Technology
9.8.2 *Mathematica* Packages
9.8.3 Java
9.8.4 Tomcat
9.8.5 Servers JSPs and Servlets
9.8.6 XML, HTML, MathML, and SVG
9.8.7 PDF Tools
9.8.8 The X Window System

©1988-2006 Wolfram Research, Inc. All rights reserved.

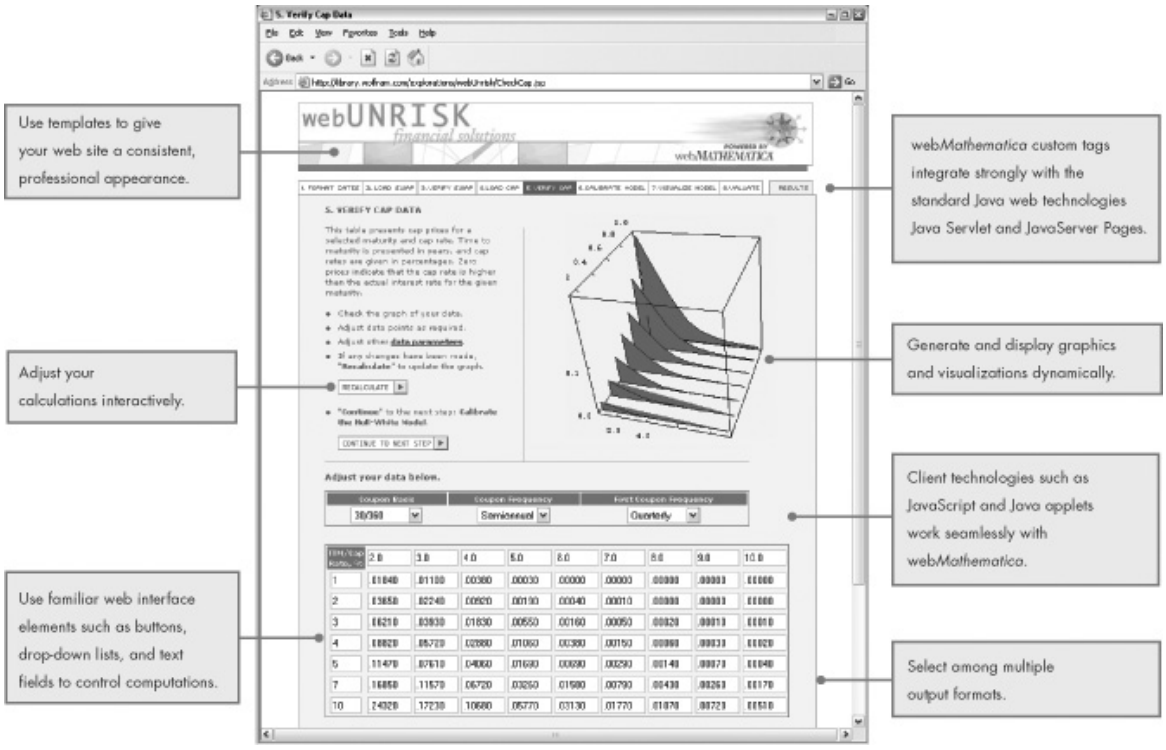
1. Introduction

This document provides a guide to the installation and operation of web*Mathematica* and the development of a web*Mathematica* site.

This introduction considers the reasons for using *Mathematica* in a web site, examines a few areas in which you might use web*Mathematica*, briefly discusses the underlying technology, and outlines the requirements for running web*Mathematica*.

1.1 What is web*Mathematica?*

web*Mathematica* adds interactive calculations and visualization to a web site by integrating *Mathematica* with the latest web server technology. The diagram below shows a view of a web*Mathematica* site, http://library.wolfram.com/explorations/webUnrisk/index.html.


Use templates to give your web site a consistent, professional appearance.
webMathematica custom tags integrate strongly with the standard Java web technologies Java Servlet and JavaServer Pages.
Generate and display graphics and visualizations dynamically.
Adjust your calculations interactively.
Client technologies such as JavaScript and Java applets work seamlessly with webMathematica.
Use familiar web interface elements such as buttons, drop-down lists, and text fields to control computations.
Select among multiple output formats.
webUNRISK
financial solutions
5. VERIFY CAP DATA
Adjust your data below.

This site gives a web browser interface to financial calculations and visualizations that are driven by *Mathematica*. In this site users are taken through a sequence of web pages in which they select different input parameters and submit data to build up a sequence of results.

1.2 Why Use *Mathematica* in a Web Site?

There are various important features that *Mathematica* can offer to a web site, including computation, an interactive programming language, connectivity, the *Mathematica* front end, and enhanced support for MathML.

©1988-2006 Wolfram Research, Inc. All rights reserved.

1.2.1 Computation

Mathematica contains a large collection of functions for computing in many areas, such as numerics, symbolics, and graphics. web*Mathematica* makes all of this functionality available over the web.

Many web technologies, so powerful in many areas, are not well suited to scientific computation; it is simply not their main focus. *Mathematica*, on the other hand, is very suitable for scientific computation and can provide this on the web.

1.2.2 Interactive Programming Language

Mathematica contains a high-level, interactive, functional programming language. It lends itself to rapid prototyping but can scale up to large intensive computations. These are also advantages for web content generation, since large sites can be developed with less programmer effort.

1.2.3 Connectivity

Mathematica connects readily to external services, which may be provided by languages such as Java, C, Fortran, or Perl. These services can provide a data source for computations and also take the results from *Mathematica*. It is particularly easy to connect to Java via *J/Link*, a toolkit for integrating Java into *Mathematica*. More information on *J/Link* can be found at http://www.wolfram.com/solutions/mathlink/jlink.

1.2.4 The *Mathematica* Front End

The *Mathematica* notebook user interface (front end) has long provided the premium mechanism for working with the *Mathematica* kernel. Now, web*Mathematica* provides an alternative interface via the web. Even in a web environment, the front end is extremely useful. It is used to typeset mathematics and render two- and three-dimensional graphical objects into images. In addition, the front end can generate notebook documents on the server to send to the client.

1.2.5 Mathematical Typesetting and MathML

Mathematica is a premium system for interactive mathematical typesetting. It is also a powerful system for working with MathML, which is designed to allow the use and reuse of mathematical and scientific content on the web and by other applications. These features are a valuable component of web*Mathematica*, which works well with the increasing number of tools that are available for MathML.

1.3 Why a Web Interface?

Some of the benefits that a web interface brings to *Mathematica* include ease of use and delivery, as well as the large number of web development professionals and the many web technologies.

©1988-2006 Wolfram Research, Inc. All rights reserved.

1.3.1 Ease of Use

To use a web*Mathematica* site, all you need is a web browser. User interfaces can use standard web GUI elements, such as text fields, check boxes, and drop-down lists. This reduces training time because users no longer have to learn different software applications. In many cases, no *Mathematica* experience is required.

1.3.2 Server-Based Configuration

There is no software to buy, install, or maintain in order to use web*Mathematica* sites. All end users need is a web browser and, for advanced features like interactive 3D graphics, a Java Runtime Environment. This leads to significant savings over buying and maintaining user software and also ensures that every end user always has the most recent version. An additional advantage is that web*Mathematica*-enhanced web sites can be accessed from many different types of computer.

1.3.3 Web Technologies

There are many people who are experts in working with servers and developing dynamic web sites. They can choose from the many web technologies and tools to develop *Mathematica*-related sites. Thus, development is easier and the applications they build are more powerful.

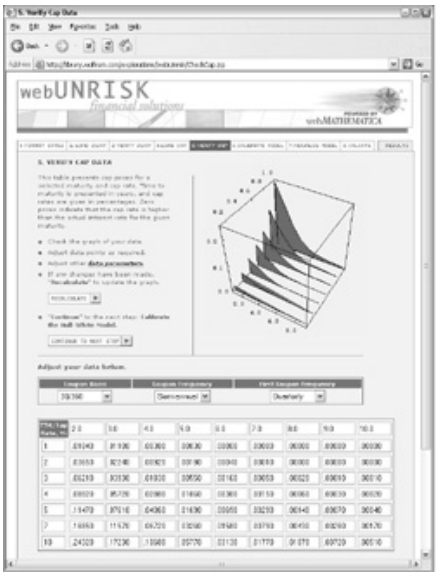
1.4 Areas of Use for web*Mathematica*

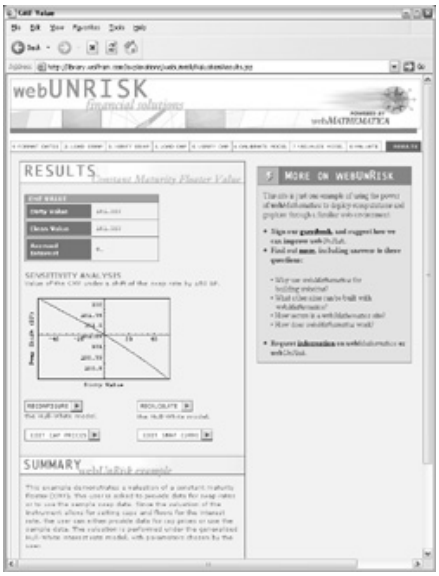
There are several areas of use for web*Mathematica*. Some of these include web computation, education, publishing, research, and hobbyist calculations.

1.4.1 Web Computation

A major use of web*Mathematica* is to build online tools for computation and visualization. An example is web*Unrisk*, http://library.wolfram.com/explorations/webUnrisk/index.html; some examples of web*Unrisk* are shown below.

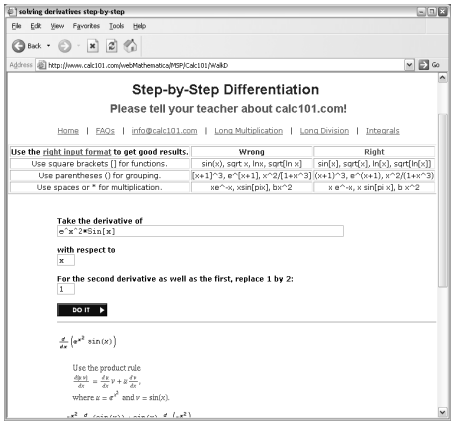
©1988-2006 Wolfram Research, Inc. All rights reserved.





1.4.2 Education

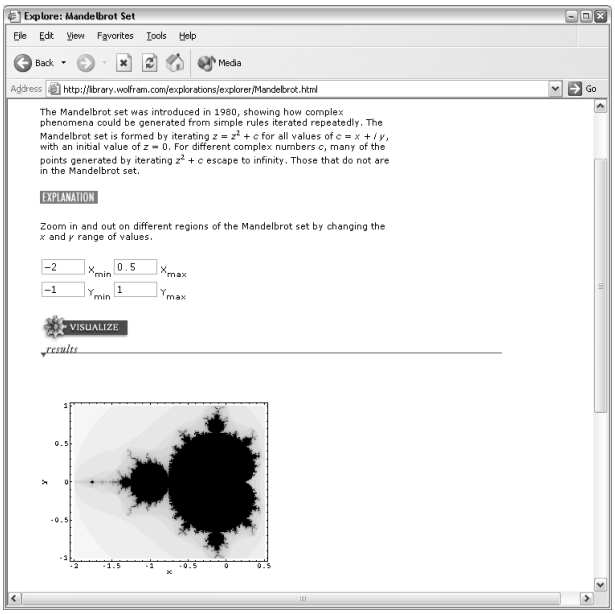
Mathematica is widely used in many areas of education. These applications can be extended to web-based education tools with web*Mathematica*. *The Integrator*, http://integrals.wolfram.com, is a Wolfram Research-developed web site that solves integration problems. Another use of web*Mathematica* in education is Calc101, http://www.calc101.com, which mixes free and pay-per-use calculators that lead precollege and college students through integration and differentiation problems, as seen here.

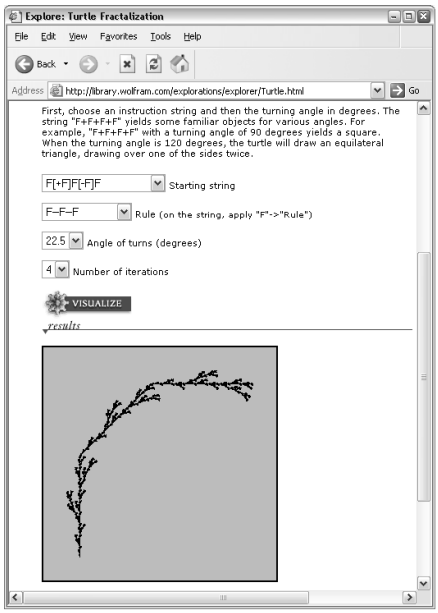


1.4.3 Publishing

Many publishers are developing web-based supplements to textbooks, manuals, and journals. web*Mathematica* provides a suitable technology to support these efforts in technical subjects. *The Mathematical Explorer*, an interactive text from Wolfram Research, covers a wide range of topics from Escher patterns and square wheels to Fermat's Last Theorem and the Riemann Hypothesis. A web-based supplement to *The Mathematical Explorer*, built with web*Mathematica*, is available at http://library.wolfram.com/explorations/explorer/index.html , as shown in the following.

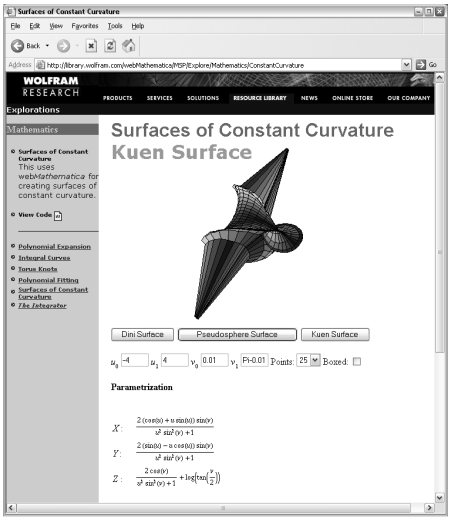
©1988-2006 Wolfram Research, Inc. All rights reserved.

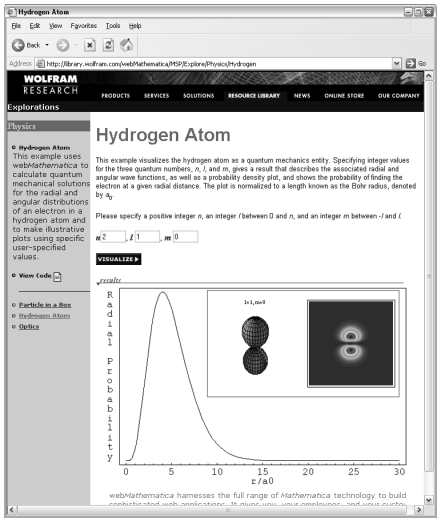

Explore: Mandelbrot Set
The Mandelbrot set was introduced in 1980, showing how complex phenomena could be generated from simple rules iterated repeatedly. The Mandelbrot set is formed by iterating $z = z^2 + c$ for all values of $c = x + iy$, with an initial value of $z = 0$. For different complex numbers c, many of the points generated by iterating $z^2 + c$ escape to infinity. Those that do not are in the Mandelbrot set.
EXPLANATION
Zoom in and out on different regions of the Mandelbrot set by changing the x and y range of values.
VISUALIZE
results


Explore: Turtle Fractalization
First, choose an instruction string and then the turning angle in degrees. The string "F+F+F+F" yields some familiar objects for various angles. For example, "F+F+F+F" with a turning angle of 90 degrees yields a square. When the turning angle is 120 degrees, the turtle will draw an equilateral triangle, drawing over one of the sides twice.
F[+F]F[-F]F
Starting string
F--F--F
Rule (on the string, apply "F"->"Rule")
22.5
Angle of turns (degrees)
4
Number of iterations
VISUALIZE
results

1.4.4 Research

Researchers all over the world use *Mathematica* to investigate their fields of interest and develop techniques and algorithms for solving problems. All the *Mathematica* work they develop can now be delivered with live interactive web sites, vastly increasing the number of people who can use and learn from their results. A typical web site that plots surfaces of constant curvature is http://library.wolfram.com/webMathematica/MSP/Explore/Mathematics/ConstantCurvature.

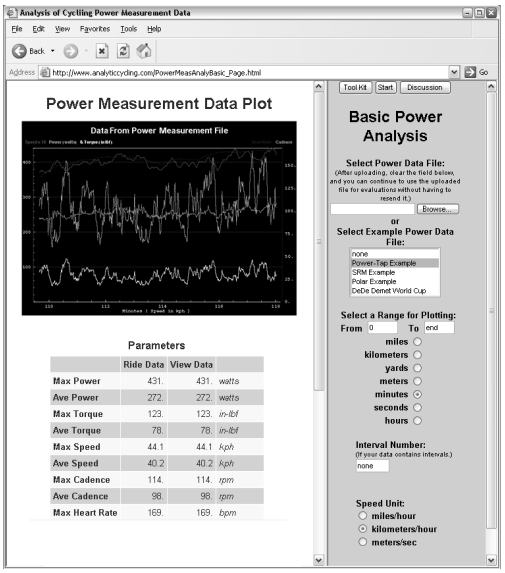


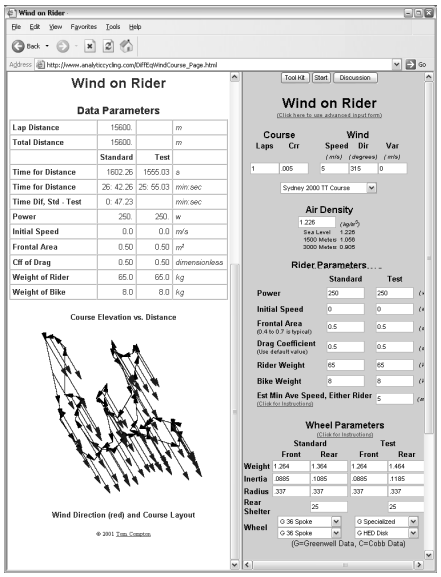


©1988-2006 Wolfram Research, Inc. All rights reserved.

1.4.5 Hobbyist

web*Mathematica* allows individual users to showcase their personal interests with web-based interactive calculations and visualizations. AnalyticCycling.com, http://www.analyticcycling.com/, is a recreational web site that takes advantage of the web*Mathematica* engine. Designed for technically oriented cyclists, AnalyticCycling.com offers web-based calculators that take a no-compromise, textbook approach to computing cycling performance.





1.5 web*Mathematica* Technology

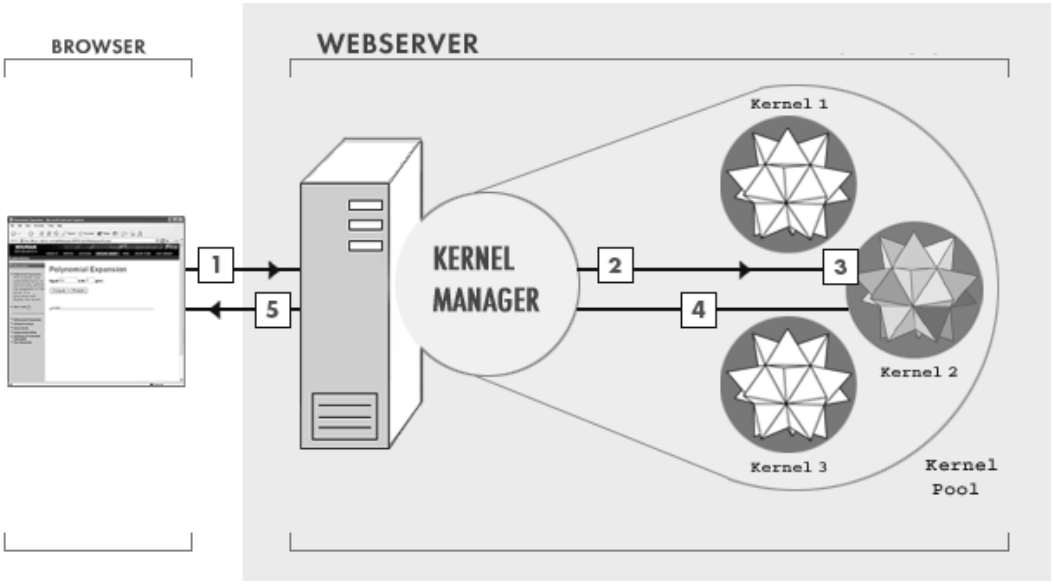
web*Mathematica* is based on two standard Java technologies: Java Servlet and JavaServer Pages (JSPs). Servlets are special Java programs that run in a Java-enabled web server, which is typically called a "servlet container" (or sometimes a "servlet engine"). There are many different types of servlet container that will run on many different operating systems and architectures. They can also be integrated into other web servers, such as the Apache web server.

web*Mathematica* allows a site to deliver HTML pages that are enhanced by the addition of *Mathematica* commands. When a request is made for one of these pages, the *Mathematica* commands are evaluated and the computed result is placed in the page. This is done with the standard Java templating mechanism, JavaServer Pages, making use of a special tags; examples of these are given in a later section.

web*Mathematica* technology uses the request/response standard followed by web servers. Input can come from HTML forms, applets, JavaScript, and web-enabled applications. It is also possible to send data files to a web*Mathematica* server for processing. Output can be many different formats such as HTML, images, *Mathematica* notebooks, MathML, SVG, XML, PostScript, and PDF. This user guide includes examples of working with all these different technologies.

web*Mathematica* provides a large library of *Mathematica* commands to handle the many possible ways of working with *Mathematica* computations. An important part of web*Mathematica* is the kernel manager which calls *Mathematica* in a robust, efficient, and secure manner. The manager maintains a pool of one or more *Mathematica* kernels and, in this way, can process more than one request at a time. An overview of the workings of a web*Mathematica* site is shown here.

©1988-2006 Wolfram Research, Inc. All rights reserved.



1 Browser sends request to web*Mathematica* server.

2 web*Mathematica* server acquires *Mathematica* kernel from the pool.

3 *Mathematica* kernel is initialized with input parameters, carries out calculations, and returns result to server.

4 web*Mathematica* server returns *Mathematica* kernel to the pool.

5 web*Mathematica* server returns result to Browser.

1.5.1 Requirements

The aim of web*Mathematica* and MSP technology is to reduce the amount of extra knowledge required for developing a site to a minimum. In practice, this means knowing something about HTML and *Mathematica*. You do not need any special knowledge of Java, nor do you need to know anything about JavaScript. web*Mathematica* also aims to automate the management of the site to make running, maintenance, and configuration as convenient as possible. Administrators of web*Mathematica* sites do not need any knowledge of Java beyond its installation.

The minimum technical components for web*Mathematica* are:

- A servlet container supporting both the Servlet Specification 2.2 (or higher) and JSP Specification 1.2 (or higher)
- A JDK 1.2 (or higher); Java 2 Version 1.4 (or higher) is recommended

There are many different combinations of hardware and operating systems that support these components. Most systems that run *Mathematica* will support web*Mathematica*. At present Intel/Windows, Intel/Linux, Sun/Solaris, IBM-RISC/AIX, PA-RISC/HP-UX, Mac OS X, AMD64/Linux, and Itanium/Linux are fully supported. For information on other platforms, check with Wolfram Research. Note that web*Mathematica* will run on many of the Microsoft operating systems such as Windows 95/98/NT/2000/XP. It is probably better to use web*Mathematica* on Windows 95/98 only for testing and demonstration purposes. Setting up the servlet container is discussed in a later section.

©1988-2006 Wolfram Research, Inc. All rights reserved.

If your servlet container only supports JSP 1.1 API you can still use web*Mathematica* as described in the section on Classic web*Mathematica* Technology.

1.6 web*Mathematica* 2.3 New Features

The main new feature of web*Mathematica* 2.3 is support for *Mathematica* 5.2. There are also a number of internal improvements.

1.7 web*Mathematica* 2.2 New Features

Support for *Mathematica* 5.1

web*Mathematica* 2.2 comes with *Mathematica* 5.1. *Mathematica* 5.1 contains many important new features relevant to web operations, the most important being optimized binary I/O, graph and array plotting, and comprehensive string manipulation, matching, and searching capabilities.

Database Connectivity

DatabaseLink provides *Mathematica* with an industrial-strength, ready-made solution for integrating *Mathematica* with any standard SQL database. Integrated with *Mathematica* 5.1 it provides a convenient bridge between SQL databases and web*Mathematica*. One particularly useful feature for web*Mathematica* is that *DatabaseLink* contains the HSQL Database Engine (HSQLDB), a lightweight database. This means that if you do not already have a database or want to experiment with using one you do not have to set one up, instead you can use HSQLDB.

Client Web Services

The *Mathematica Web Services Package* allows *Mathematica* to call web services across the internet. Bundled with *Mathematica* 5.1, it provides a convenient way for web*Mathematica* to use a web service. This is an important way to extend the functionality of a web*Mathematica* web site.

Authoring System

web*Mathematica* Author is an authoring system that aids in the creation and development of web*Mathematica* material. A preview version is shipped with web*Mathematica* 2.2.

1.8 web*Mathematica* 2.1 New Features

The main new feature of web*Mathematica* 2.1 is support for *Mathematica* 5.0. There are also a number of internal improvements and new examples.

1.9 web*Mathematica* 2.0 New Features

web*Mathematica* 2.0 offered a number of new features and improvements. These are listed in this section.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Support for *Mathematica* 4.2

web*Mathematica* 2.0 comes with *Mathematica* 4.2. *Mathematica* 4.2 has many features that are very relevant to web operations, the most important being the XML support. There are many examples in web*Mathematica* 2.0 that use XML features and XML applications such as MathML and SVG.

Simplified Installation

web*Mathematica* 2.0 has a simplified installation process that only requires the installation of the web*Mathematica* web application. There is a minimum of extra configuration that is required.

Extended Documentation and Examples

The documentation for web*Mathematica* is now shipped in HTML format and accessible from the web*Mathematica* front page. In addition many new examples have been added that demonstrate the new features.

New Templating Mechanism Based on JSP Custom Tags

A new HTML templating mechanism based on JSP custom tags has been added. This is now the preferred mechanism for using web*Mathematica*. The mechanism is easier to understand, it allows the use of other JSP custom tag libraries, and it facilitates the integration of web*Mathematica* into other server applications.

MathML, SVG, and XML Support

Support for the XML applications, MathML and SVG, is built into web*Mathematica* 2.0. In addition it can make use of the new XML processing tools that are available in *Mathematica* 4.2.

Support for Catching Message and Print Output

New functions are provided for catching the output of any *Mathematica* Message or Print statements. This can be useful for debugging or developing material.

Support for HTTP File Upload

New functions are provided to support HTTP file upload. This is an important way to submit information to a web*Mathematica* web server.

Support for HTTP Session Variables

New functions are provided for saving material in an HTTP session stored in the server. This can be useful for saving results from one computation to another.

©1988-2006 Wolfram Research, Inc. All rights reserved.

HTML Formatting Functions

New functions are provided for formatting results into HTML.

1.9.1 Modifications in web*Mathematica* 2.0

This section lists any changes in web*Mathematica* 2.0 that work differently from 1.0.

Location of Security Configuration File

The mechanism for locating the security configuration file has changed from web*Mathematica* 1.0. Now the security configuration file is named in the pool configuration file and is located in a central configuration directory in `webMathematica/WEB-INF/conf`. Previously the configuration file could be loaded from anywhere on the *Mathematica* path.

This change was made because loading the security configuration from a single central location is more secure. Since the default security system of web*Mathematica* is very conservative, any sites that do not move their security files will run with a higher level of security than is expected. This is discussed in the Security section.

Location of MSP.conf

The default location MSP.conf has been moved into a central configuration directory in `webMathematica/WEB-INF/conf`. This leads to a great simplification in the setup of your server because it will look automatically in this location.

©1988-2006 Wolfram Research, Inc. All rights reserved.

2. Installation

These installation instructions focus on setting up a servlet container and installing web*Mathematica*. As explained previously, web*Mathematica* is based on a standard Java technology called servlets; support for servlets is typically provided by a program called a servlet container. You must set up the servlet container before adding web*Mathematica*.

The flow of installation is as follows:

- Set up a servlet container.
- Set up *Mathematica* using the *Mathematica* CD-ROM from your distribution.
- Install the web*Mathematica* web application into your servlet container using the web*Mathematica* Tools CD-ROM from your distribution.
- For Unix, you may need to set up an X server.
- Finally, you will test your web*Mathematica* site.

More information on installation of web*Mathematica* can be obtained from Wolfram Research at http://documents.wolfram.com/webmathematica/updates/install.

2.1 Setting up a Servlet Container

Before you start to install web*Mathematica,* you need an installation of Java and a servlet container. If you already have these components, you may skip this section.

There are many different servlet containers, but one that is particularly convenient is Tomcat, which can be obtained from http://jakarta.apache.org. Another popular server container is JRun, http://www.macromedia.com/software/jrun/. Since Tomcat is a common way to run web*Mathematica*, we have added information on installing and setting it up on Unix and Windows.

web*Mathematica* has been tested with both Tomcat and JRun as well as other containers listed at http://www.wolfram.com/products/webmathematica/technology/specifications.html . If you have a particular interest or experience in running web*Mathematica* with other containers, please contact Wolfram Research. However, if you do not have expertise with these other containers, we recommend you use Apache Tomcat.

When your servlet container is functioning correctly, as demonstrated by running its sample servlets, you are ready to install web*Mathematica*. If your servlet container does not work, then web*Mathematica* cannot work. The remaining steps in this section show you how to set up Java and Tomcat. If you are not using Tomcat, you should skip this section and study the documentation for your servlet container.

©1988-2006 Wolfram Research, Inc. All rights reserved.

2.1.1 Setting up Java

Any Java Development Kit (JDK) that supports Java Version 1.2 (or later) can be used. However, it is recommended that you use a modern version of Java, such as Java 2 Version 1.4. For Windows NT/2000/XP, Linux, or Solaris, these are available from the Sun Java site at http://java.sun.com/j2se/downloads/index.html. The Sun Java site provides detailed installation instructions for the different platforms. These are all relatively simple; typically, you download and execute an installer. To run web*Mathematica* you need the JDK. For Mac OS X, Java 2 Version 1.3 is already installed. For other platforms, modern versions of Java are available from the appropriate vendors; a list of useful links is maintained in the Appendix: Java.

You will also need to set the JAVA_HOME environment variable. This is described in the next sections for Unix and Windows.

Unix

The JAVA_HOME environment variable needs to be set for the environment in which Tomcat runs. An example of this, suitable for inclusion in .bashrc (this is the initialization file for the bash shell), is shown below.

```
JAVA_HOME=/usr/local/j2sdk1.4
export JAVA_HOME
```

For other shells, you should follow their standards for setting environment variables.

Windows

It is recommended to set the JAVA_HOME environment variable for the environment in which Tomcat runs. This can be done with a script used to launch Tomcat or more conveniently with the Control Panel.

If you go to the Control Panel and open the System icon, you will see the System Properties window. From this, select the Advanced tab and then the Environment Variables button. Enter JAVA_HOME as a system variable, setting it to the top-level directory containing your JDK. For example, if your JDK is installed in c:\j2sdk, this is the setting for JAVA_HOME.

It can also be useful to amend the PATH environment variable to include the directory that contains the jawt.dll library. Typically this is found in JAVA_HOME/jre/bin.

Mac OS X

Mac OS X 10.3 ships with Java 2 Version 1.4.1. You may find that an updated version can be obtained via the software update mechanism (see also http://www.apple.com/java/). If you update your Java, you can ensure that you are always using the most recent version of the JDK by setting up the JAVA_HOME environment variable properly; this is shown below.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
JAVA_HOME=/System/Library/Frameworks/JavaVM.framework/Home
export JAVA_HOME
```

The default login shell for Mac OS X 10.3 is `bash`; hence, the above command needs to be placed in the appropriate shell initialization file, for example, `.bashrc`.

Mac OS X 10.2 was originally shipped with Java 2 Version 1.3.1, but updates are available from the software update mechanism (see also http://www.apple.com/java/). If you update your Java, you can ensure that you are always using the most recent version of the JDK by setting up the `JAVA_HOME` environment variable properly; this is shown below.

```
setenv JAVA_HOME /System/Library/Frameworks/JavaVM.framework/Home
```

The default login shell for Mac OS X 10.2 is `tcsh`; hence, the above command needs to be placed in the appropriate shell initialization file, for example, `.tcshrc`.

2.1.2 Setting up Tomcat

This section describes setting up Tomcat on Unix and Windows. The main web site for Tomcat is http://jakarta.apache.org; a list of useful links is maintained in Appendix: Tomcat.

Unix

Before you run Tomcat, you should first make sure you have set up a JDK on your machine; this was described in the previous section.

Download information for current versions of Tomcat is given at http://documents.wolfram.com/webmathematica/updates/install/tomcat/current. A variety of archive formats are available; one of these should be unpacked in some central location, for example `/usr/local`. You may also wish to change the name of the top-level directory. The actual location of Tomcat and the name of the top-level directory are entirely up to you. Sample shell instructions for these steps are shown below (note that `tar xvfz archive` will give you more information on what files are being extracted). Other versions of Tomcat are available from the Apache web site, http://jakarta.apache.org.

```
[server1]$ cd /usr/local
[server1]$ tar xfz jakarta-tomcat-5.0.28.tar.gz
[server1]$ mv jakarta-tomcat-5.0.28 tomcat
```

On some platforms, such as Solaris, the default `tar` command does not work to unpack the Tomcat archive as shown above. One would need to obtain the GNU tar utility from http://www.gnu.org/directory/GNU/tar.html in order to use the options shown.

It is often useful to create a low privilege account, such as `tomcat`, to run your servlet container. It is probably helpful if this account has a home directory so that your X server and *Mathematica* can store preferences informa-

©1988-2006 Wolfram Research, Inc. All rights reserved.

tion. If you create such an account, you may need to change ownership of the Tomcat layout so it can be run by this account.

```
[server1]$ chown -R tomcat tomcat
```

The main top-level directory of Tomcat contains some important directories, including:

```
tomcat
    bin
    conf
    logs
    webapps
```

The `bin` directory contains commands for running Tomcat; the `conf` directory contains site configuration files; the `logs` directory contains various log files; the `webapps` directory is where you will install web*Mathematica*. You should be able to launch Tomcat immediately from the `bin` directory, making sure to be the `tomcat` user.

```
[server1]$ su tomcat
[server1]$ cd tomcat/bin
[server1]$ ./startup.sh
```

At this point, you should be able to connect to Tomcat via a URL such as http://localhost:8080. If this does not return the Tomcat front page, then something is wrong with your setup. If you look at the log files, it may help you track down your problem. Make sure that you have set your `JAVA_HOME` variable as described in the installing Java for Unix section.

The `bin` directory also contains a script, `shutdown.sh`, used for shutting down Tomcat.

Information on launching Tomcat automatically on Unix is given in a later section.

Windows

Before you run Tomcat, you should first make sure you have set up a JDK on your machine; this was described in the previous section.

Download information for current versions of Tomcat is given at http://documents.wolfram.com/webmathematica/updates/install/tomcat/current . A convenient way to install Tomcat is to download the self-installing executable. You should launch the installer and follow the instructions it provides. If you choose not to use the self-installing executable, then unpack the binary distribution into a convenient location. Other versions of Tomcat are available from the Apache web site, http://jakarta.apache.org.

After installation is complete, you may wish to inspect the main top-level directory of Tomcat, which contains some important directories, including:

```
tomcat
    bin
    conf
    logs
    webapps
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

The bin directory contains commands for running Tomcat; the conf directory contains site configuration files; the logs directory contains various log files; the webapps directory is where you will install web*Mathematica*.

The installer adds a Start Menu Group from which you can run Tomcat. You should test it via a URL such as http://localhost:8080. If Tomcat does not run correctly, you should open a command prompt window, change directories ("cd") to the bin directory (in the main top-level directory of Tomcat) and try running the startup.bat batch file (this can also be accomplished by double-clicking on the file via the Windows Explorer). Starting and stopping Tomcat from the Start Menu is very convenient (this is also a new feature of Tomcat 4.1), but for running Tomcat as a production server under Windows you may wish to run it as a Windows Service. This is described in the section on launching Tomcat automatically on Windows.

Mac OS X

Of course, before you run Tomcat, you should first make sure you have set up a JDK on your machine as described in the previous section.

Mac OS X 10.3 already includes a version of Tomcat, which is located in /Library/Tomcat. Therefore, there is no need to obtain and install Tomcat.

It is often useful to create a low privilege account, such as tomcat, to run your servlet container. You can accomplish this via the System Preferences panel. If you create such an account, you may need to change ownership of the Tomcat layout so it can be run by this account.

```
[server1]$ sudo chown -R tomcat tomcat
```

The main top-level directory of Tomcat contains some important directories, including:

```
tomcat
    bin
    conf
    logs
    webapps
```

The bin directory contains commands for running Tomcat; the conf directory contains site configuration files; the logs directory contains various log files; the webapps directory is where you will install web*Mathematica*. You should be able to launch Tomcat immediately from the bin directory, making sure to be the tomcat user.

```
[server1]$ su Tomcat
[server1]$ cd Tomcat/bin
[server1]$ ./startup.sh
```

At this point, you should be able to connect to Tomcat via a URL such as http://localhost:8080. If this does not return the Tomcat front page, then something is wrong with your setup. If you look at the log files, it may help you track down your problem. Make sure that you have set your JAVA_HOME variable as described in the installing Java for Mac OS X section.

The bin directory also contains a script, shutdown.sh, used for shutting down Tomcat.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Please also note that for web*Mathematica* to fully function, you need to log on via the Mac OS X console. This is necessary since the *Mathematica* front end makes use of the Mac OS X windowing environment.

Tomcat is not already installed on versions of Mac OS X older than 10.3. Download information for current versions of Tomcat is given at http://documents.wolfram.com/webmathematica/updates/install/tomcat/current . A variety of archive formats are available; one of these should be unpacked in some central location, for example, `/Library`. You may also wish to change the name of the top-level Tomcat directory. The actual location of Tomcat and the name of the top-level directory are entirely up to you. `/Library` is useful because it can be viewed via the `Finder`. Other versions of Tomcat are available from the Apache web site, http://jakarta.apache.org.

Note that the default OS X `tar` command does not work to unpack the Tomcat archive as shown below. You would need to use the GNU `tar` utility (`gnutar`), which normally resides in `/usr/bin/`, to use the options shown. You could also use Stuffit Expander (Version 7.0.1 and later) which uncompresses `*.tar.gz` archives.

Sample shell instructions for these steps are shown below (note that `tar xvfz` archive will give you more information on what files are being extracted). These instructions assume that you are using the `Terminal` application found in `Applications-> Utilities-> Terminal`.

```
[server1]$ cd /Library
[server1]$ sudo /usr/bin/gnutar xfz jakarta-tomcat-5.0.28.tar.gz
[server1]$ sudo mv jakarta-tomcat-5.0.28 tomcat
```

2.2 Installing and Configuring *Mathematica*

Install *Mathematica* from the *Mathematica* CD-ROM using the *Getting Started with Mathematica* manual appropriate for the platform you wish to use for your web server. You should choose a single-machine installation and install an appropriate license as described in the manual. When you have finished, you should be able to run *Mathematica* interactively to validate your installation. If *Mathematica* cannot run, then web*Mathematica* cannot run.

If you already have an installation of *Mathematica* on your server, you do not need to install *Mathematica* again, but can proceed with the remaining installation steps. With an existing installation of *Mathematica*, you may place your web*Mathematica* license information into a different location, as described in the following section. Placing the license information in a different location will ensure that an interactive usage of *Mathematica* on your server does not interfere with the operation of your web*Mathematica* site. One possible alternative directory is the `web Mathematica/WEB-INF/conf` directory as demonstrated in the section on installing web*Mathematica* into Tomcat. Note that if you install the license in a special file, you will have to set the `-pwfile` option when you run *Mathematica* outside of web*Mathematica*.

©1988-2006 Wolfram Research, Inc. All rights reserved.

2.3 Installing the web*Mathematica* Web Application

This section describes how to install web*Mathematica* components into your servlet container. For most servlet containers, this involves deploying the web*Mathematica* web application (webapp) found in archived form on the web*Mathematica* Tools CD-ROM. Separate installation instructions are given for some different servlet containers. A web application is a collection of HTML and other web components, which are placed in a specific directory structure. Any servlet container that supports web applications will be able to use these files in a standard way. Web applications support a special type of archive called a WAR archive, which is supported by some servlet containers. The web*Mathematica* Tools CD-ROM provides a WAR archive of the web*Mathematica* archive; the contents of the archive is given in Appendix: File Layout.

2.3.1 If You Use Tomcat

This section describes the deployment of the web*Mathematica* webapp in Tomcat. There are two steps: unpacking the web*Mathematica* archive and configuring the `MSP.conf` file.

First, choose one of the web*Mathematica* archives from the Tools CD-ROM; for example, `webMathematica.zip` or `webMathematica.tar.gz`. Unpack the archive into the web*Mathematica* directory located in the Tomcat `webapps` directory. This is usually found in the top-level directory of Tomcat. You have now created a web application called web*Mathematica*. The contents of the web*Mathematica* web application are described later in Appendix: File Layout. Some of the contents of the top directory of Tomcat, along with the location of the `webapps` directory and web*Mathematica* web application, are shown below.

```
tomcat
    conf
    bin
    logs
    lib
    webapps
        webMathematica
```

Second, configure the file `MSP.conf`, located in the `WEB-INF/conf` directory. This file holds various site-specific parameters and may need modification for your site. The settings that can be placed into `MSP.conf` are described in the section Site Configuration.

The most important setting is `MathLinkArguments`, the command that launches *Mathematica*. The `MSP.conf` that ships with web*Mathematica* contains settings suitable for a default installation of *Mathematica* for Windows, Unix, Linux, and Mac OS X. However, if you install *Mathematica* into a nondefault location, you will need to modify this file. For example, if you installed *Mathematica* into `E:\Mathematica`, make the following setting of `MathLinkArguments` in `MSP.conf` (note the backslashes).

```
MathLinkArguments=-linkname 'E:\\Mathematica\\MathKernel.exe
    -mathlink' -linkmode launch
```

Another reason to modify `MSP.conf` is to store your web*Mathematica* license in its own password file, for example, `webMathematica/WEB-INF/conf/mathpass`. It would then be necessary to modify `MSP.conf` to ensure that *Mathematica* uses this location. The following shows how this could be done for a typical Windows installation.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
MathLinkArguments=-linkname  'c:\\Program Files\\Wolfram
    Research\\Mathematica\\5.2\\MathKernel.exe -mathlink -pwfile
    "c:/Program
    Files/tomcat/webapps/webMathematica/WEB-INF/conf/mathpass"'
    -linkmode launch
```

A typical setting for `MSP.conf` to use a special `mathpass` file under Unix is shown below.

```
MathLinkArguments=-linkname  'math -mathlink -pwfile
    /usr/local/tomcat/webapps/webMathematica/WEF-INF/conf/mathpass'
    -linkmode launch
```

Under Unix, you may need to add a `FrontEndLaunchFlags` parameter so the front end can run properly. In the following example, the front end will be launched to use `DISPLAY 1` with fixed geometry and in server mode. For more information on running the front end under Unix, see the documentation section Configuring for the X Window System.

```
FrontEndLaunchFlags=-display :1
```

2.3.2 If You Do Not Use Tomcat

JRun

JRun is a sophisticated application server. It contains a web-based wizard to guide you through the installation process. You will need to follow JRun-specific instructions to install the web*Mathematica* web application that are given in this section. Please note that you need to use JRun 4.0 or higher in order to use the web*Mathematica* custom tags. If you use JRun 3.x, you can still use web*Mathematica* as described in the section Classic web*Mathematica* Technology. These installation instructions are appropriate for JRun 4.

First, unpack the web*Mathematica* archive into a location on your server that is accessible by JRun, e.g. `E:\web Mathematica`. At this time you can make any necessary changes to `MSP.conf`. For example, you may wish to set the `MathLinkArguments` or the `FrontEndLaunchFlags` as described in the section for installing into Tomcat.

Now use the `JRun Management Console`, which under Windows is accessible via `Start->Programs-> Macromedia JRun4->JRun Management Console`. Alternatively, you can use the URL http://localhost:8000. You will need to enter the username and password that were specified when JRun was installed. Once you gain access, click the `default` link on the left frame to expand its contents. In the main frame, you should see information about the default server, which has the title `J2EE Components on default`.

Click the `add` button under `Web Applications` to add the web*Mathematica* web application. You will see a new page, entitled `Deploy Web Application`, with a field called `Source File Path`. Click on the `Browse` button - this will bring up a file browsing window. Navigate to the location (e.g. `E:\webMathematica`) where you have extracted the files from the web*Mathematica* archive. Select the web*Mathematica* folder icon

©1988-2006 Wolfram Research, Inc. All rights reserved.

and click the Open button. This will change the Source File Path to E:\webMathematica. You can now click Deploy. After a few moments, you will see a screen that allows you to customize various details of the installation, depending on the version of JRun that you are using. You can choose the default values and then click Apply. A message informing you that the deployment of web*Mathematica* has been successful will appear.

At this point you should be able to test your installation as described in the Testing section. Note that the default port for JRun is 8100, so a URL such as http://localhost:8100/webMathematica, should access the web*Mathematica* front page.

Other Servlet Engines

If you are unfamiliar with servlets, then it is recommended that you use Apache Tomcat. You should only use another servlet engine if you are already experienced with it.

If you have some other servlet engine, follow its instructions for installing a web application, which may be supported by some special tools. After installing the web application, you will need to modify the MSP.conf file as described in the section on installing under Tomcat. It may also be necessary to make various modifications to MSP.conf, such as changing the WebApplicationName or the JLinkNativeLibraryDirectory settings. These are described in Appendix: MSP.conf.

2.4 Configuring for the X Window System (Unix only)

There are special problems associated with running the *Mathematica* front end under X from within a web server. This is because, typically, the web server is run as a special account, such as tomcat. This means that when web*Mathematica* runs the front end, it is running as this account. For the front end to operate, it must connect to an X server; this could be achieved by logging into the console of the web server machine with this special account running an X server. There are a few problems with this approach: first, you may not want to leave the machine with an open login on the console; secondly, every time the front end does something, a window will appear on the screen, which may be distracting for someone using the machine.

If a different user logs into the console and runs an X server, the front end (which is run by the special web*Mathematica* account) will not be able to connect to this server at all under the standard authentication system of X. While it is possible to configure the server to allow these connections, it is not satisfactory because web*Mathematica* will be displaying windows on the screen every time it does something with the front end. This topic is discussed in greater length in a later section.

These problems are solved by running a virtual X server, such as Xvnc, as described in the following section. Running a virtual server prevents the windows created by the *Mathematica* front end from displaying on the screen console of the computer running *Mathematica*.

2.4.1 Configuring Xvnc and web*Mathematica*

Xvnc is the Unix VNC server. It provides a virtual X server that can be used by applications, such as the *Mathematica* front end, when it is running for web*Mathematica*. It also provides a VNC server so that a VNC viewer can connect to view and control any applications running in the server. This can be useful since it can help track down problems in running the front end.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Distributions of Xvnc are available from RealVNC, http://www.realvnc.com, and TightVNC, http://www.tightvnc.com. It should also be noted that certain Linux distributions contain Xvnc in RPM package format.

Install Xvnc

Installation of Xvnc is quite straightforward; you unpack the archive and then copy the relevant files into some local `bin` directory, e.g. `/usr/local/bin`. (Note that some modern Linux distributions already have `Xvnc` installed.) Copying of the `Xvnc` binaries is shown below.

```
[server1]$ cp vncviewer vncserver vncpasswd vncconnect Xvnc
    /usr/local/bin
```

Most Xvnc archives have an install script and instructions for installation.

Launch Xvnc

Launching Xvnc is also quite straightforward; this should be done as the user that will be running web*Mathematica*, for example, the user `tomcat`. The first time Xvnc is launched, it asks for a password. In the example below, the server is launched as `display :1`.

```
[server1]$ su tomcat
[server1]$ vncserver
You will require a password to access your desktops.
Password:
Verify:
New'X' desktop is server1:1
Creating default startup script /home/tomcat/.vnc/xstartup
Starting applications specified in /home/tomcat/.vnc/xstartup
Log file is /home/tomcat/.vnc/server1.wolfram.com:1.log
```

Test Xvnc

Once it is launched, it is a good idea to test the server by running an application to use it, for example, the *Mathematica* front end. This is shown below.

```
[server1]$ su tomcat
[server1]$ mathematica -display :1
```

Of course, when this is done, you do not see an actual window on your screen; this is because it is a virtual server. To see the window from the *Mathematica* front end, you can run the `vncviewer`. This can be done as shown below.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
[server1]$ vncviewer :1
```

This should show you a screen with the front end window visible. You should be able to type into the front end and see it working. At any time web*Mathematica* is running, you can view the desktop with `vncviewer`.

Configure web*Mathematica*

The final step is to modify your `MSP.conf` file to instruct the front end to use this server. The setting of `FrontEndLaunchFlags` is described in the Appendix: Site Configuration. Here is a sample setting that connects to the X server on `display :1`. (This setting is present in `MSP.conf` as a comment.)

```
FrontEndLaunchFlags=-display :1
```

Now your Unix server should be ready to run web*Mathematica*.

If you find that the front end does not launch correctly, it may help to add the name of the server in the configuration file. An example is shown below; here, `myserver` is the name of the machine on which web*Mathematica* and `Xvnc` are running.

```
FrontEndLaunchFlags=-display myserver:1
```

2.4.2 Other X Related Issues

The following section describes a number of further issues that relate to using web*Mathematica* in conjunction with an X server. If you have set up an Xvnc server as described above it should not be necessary to study these.

Connecting to the X Server

When the *Mathematica* front end runs, it must connect to an X server. If the X server is being run by a different user than that which is running the front end, the X server will reject the connection, as shown below.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
[root]# su tomcat
bash$ mathematica
Xlib:connection to ":0.0" refused by server
Xlib:Client is not authorized to connect to Server
xset:unable to open display ":0.0"
Xlib:connection to ":0.0" refused by server
Xlib:Client is not authorized to connect to Server
xset:unable to open display ":0.0"
Xlib:connection to ":0.0" refused by server
Xlib:Client is not authorized to connect to Server
xset:unable to open display ":0.0"
Xlib:connection to ":0.0" refused by server
Xlib:Client is not authorized to connect to Server
XMathematica:can't open display:0.0,exiting...
bash$
```

One way to avoid this problem is to allow all connections from the local machine.

```
[root]# xhost +localhost
localhost being added to access control list
[root]# su tomcat
bash$ mathematica
bash$
```

This is not a good technique since there is a potential for security problems. These are probably limited since it is only connections from the same machine that are allowed. Another problem is that every time the front end is used, a window will be drawn on the screen, which may be annoying to a user of the system.

A more satisfactory alternative is to run a virtual X server, such as Xvnc.

Xvfb

Xvfb is a virtual frame buffer server described at http://www.xfree86.org/4.3.0/Xvfb.1.html. It can be used as an alternative to Xvnc, but typically we have found Xvnc to be easier to use and provide more functionality.

For Linux, you can download an RPM archive from http://www.redhat.com. After installation, you can launch it as follows (you will probably run this as `root`).

```
su tomcat -c "/usr/X11R6/bin/Xvfb :1 -fp
unix/:7100,/usr/local/Wolfram/Mathematica/4.2/SystemFiles/Fonts/Type1,
/usr/local/Wolfram/Mathematica/4.2/SystemFiles/Fonts/BDF -screen 0
    800x600x24 " &
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

This command launches Xvfb referencing a font server on port 7100 and adding directories that contain the *Mathematica* fonts. Note that if you install *Mathematica* in some alternative location, you should modify these directories. Under some *Mathematica* installations the location of *Mathematica* fonts is added to the font server configuration; in this case the *Mathematica* fonts do not need to be referenced when Xvfb is launched. Xvfb could then be launched as shown here.

```
su tomcat -c "/usr/X11R6/bin/Xvfb :1 -fp unix/:7100" &
```

In these examples, Xvfb expects to use port 7100 on the local machine for the font server. The actual setting may need to be modified if some alternative configuration of font server is used. For example, under Redhat 6 the font server uses a local Unix socket and Xvfb should be launched as follows.

```
su tomcat -c "/usr/X11R6/bin/Xvfb :1 -fp
unix/:-1,/usr/local/Wolfram/Mathematica/4.2/SystemFiles/Fonts/Type1,
/usr/local/Wolfram/Mathematica/4.2/SystemFiles/Fonts/BDF -screen 0
    800x600x24 " &
```

If you are not running a font server, you may need to launch Xvfb with no font server reference. In this case, it may be necessary to copy the *Mathematica* fonts into the X distribution layout as described in the section below on manual font installation.

```
su tomcat -c "/usr/X11R6/bin/Xvfb :1 -screen 0 800x600x24 "&
```

Once you have launched the virtual frame buffer server, you can test that it is running. You will probably run this as root.

```
[root]# su tomcat
bash$ mathematica -display :1
```

Of course, one problem with confirming that the front end is running correctly with this server is that you cannot see it on the screen! This makes it hard to see a dialog box indicating an error. One way to see what the front end is displaying is to inspect a dump of the server with xwd and xwud, which you can do with the following.

```
xwd  -display :1 -root | xwud
```

This will show what the front end is displaying. For example, if you see a message about not finding the password, you may need to add a pwfile command-line option.

When you are running the virtual frame buffer X server, you will need to modify your MSP.conf file to instruct the front end to use this server. The setting of FrontEndLaunchFlags is described in Appendix: Configuration. Here is a sample setting.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
FrontEndLaunchFlags= -mathlink -display :1 -nogui -geometry
    1000x500+10+10
```

On some systems, such as Sun/Solaris, the X server has problems when being launched by the user `tomcat` since the permissions to the `/tmp/.X11` directories have been restricted for security reasons. The problem manifests itself with a message that says the system cannot establish any listening sockets. One solution would be to modify the directories so that `tomcat` can write to them. For more detail, see http://www.faqs.org/faqs/Solaris2/FAQ/.

Manual Font Installation

The front end cannot run without access to the *Mathematica* fonts. If you notice from the output of the X server with `vncviewer` or `xwd` that the front end is displaying a dialog box indicating that it cannot find its fonts, you will have to take some further steps to locate the fonts. One solution that is simple but drastic is to copy the *Mathematica* fonts into your X distribution.

```
cd /usr/X11R6/lib/X11/fonts
cp -r 75dpi 75dpi.orig
cd 75dpi
cp /usr/local/Wolfram/Mathematica/5.2/SystemFiles/Fonts/X/*.bdf .
mkfontdir
```

This is really a poor solution to be avoided if possible. One deficiency is that if you update your copy of *Mathematica*, you will have to remember to copy the new fonts. The proper solution is to launch `Xvfb` so it either uses a font server or a font path setting, as described above. Remember that this is not a problem when working under Windows.

2.5 Upgrading from web*Mathematica* 2.2

This section discusses some of the issues that will concern you if you already have web*Mathematica*. If you are using an older servlet container, this may be a good opportunity to upgrade to something more recent. If you are going to upgrade your servlet container, you could follow the instructions at the beginning of this chapter as though this was a fresh installation of web*Mathematica*.

2.5.1 Install *Mathematica*

web*Mathematica* comes with a copy of *Mathematica* and this should be installed as discussed in the section on Installing and Configuring Mathematica.

If you have installed any applications into your copy of *Mathematica*, you will need to make them available to *Mathematica*. This is discussed in the section Installing Packages. Note that you should not copy the MSP application to *Mathematica*.

©1988-2006 Wolfram Research, Inc. All rights reserved.

2.5.2 Install the web*Mathematica* Web Application

To install the web*Mathematica* web application, you will need to remove your current web*Mathematica* web application out of your servlet container. For Tomcat, this is a simple matter of moving the `webMathematica` directory from the `webapps` directory. You may need some of the material in this web application, so you should probably keep it somewhere accessible. After this you can follow the instructions for Installing the webMathematica Web Application.

2.5.3 Configure the New Layout

If you made any special configuration to your old version of web*Mathematica*, you may want to make similar changes in web*Mathematica* 2.3. Two relevant files are `web.xml` and `MSP.conf`. In addition, the security system is configured in a different way.

web.xml

You should only make changes to `web.xml` if you are certain that you understand the intent of the setting. In addition, it would be better to copy any modified configuration parameters from the old version rather than taking the entire file. The `web.xml` file is found in the directory `webMathematica/WEB-INF`.

MSP.conf

In web*Mathematica* `MSP.conf` is found in the directory `webMathematica/WEB-INF/conf`. You should copy specific configuration parameters rather than taking the entire file. `MSP.conf` contains the location of *Mathematica*, so you should make sure that it refers to the *Mathematica* installed earlier.

Security Configuration

The mechanism for locating the security configuration file has changed from web*Mathematica* 1.0. Now the security configuration file is named in the pool configuration file and is located in a central configuration directory in `webMathematica/WEB-INF/conf`. Previously the configuration file could be loaded from anywhere on the *Mathematica* path.

This change was made because loading the security configuration from a single central location is more secure. Since the default security system of web*Mathematica* is very conservative, any sites that do not move their security files will run with a higher level of security than is expected. Security is discussed in a later section.

2.5.4 Move Content to the New Layout

Now you should copy your content from your old layout. You should not copy any of the examples because web*Mathematica* 2.3 has its own set of updated examples. Instead, you should just copy your own material from the old layout into the new.

©1988-2006 Wolfram Research, Inc. All rights reserved.

2.5.5 Finalize the Installation

When you have installed the new version of web*Mathematica* you should test your installation. A good URL to use is http://localhost:8080/webMathematica/Examples/Specification.jsp. This will print the version numbers of your installation. You should confirm that you have web*Mathematica* 2.3 and *Mathematica* 5.2.

2.6 Optional Further Configuring

There are a number of additional features that can be obtained by optional extra installation steps. It is not necessary to carry them out to run your web*Mathematica* server, but they are included in this section to provide extra information.

2.6.1 MSP *Mathematica* Application

The MSP *Mathematica* application contains various *Mathematica* packages and utilities such as documentation. It is already installed in the web*Mathematica* layout so that web*Mathematica* will operate without any extra steps. However, you may wish to use some of the functions or to view the web*Mathematica* documentation in the *Mathematica* Help Browser. In order to do this, you will need to install the application. This installation step is optional because it is not needed for web*Mathematica* to run. The contents of the archive is given in Appendix: File Layout.

The MSP *Mathematica* application is in archive form on the web*Mathematica* Tools CD-ROM. Unpack the appropriate archive and place its contents into the `AddOns/Applications` directory in your *Mathematica* directory. It should then be possible to launch *Mathematica* and choose Rebuild Help Index from the Help menu. This will cause the web*Mathematica* documentation to appear in the Help Browser under the Add-ons category.

The MSP *Mathematica* application contains useful functions for formatting *Mathematica* expressions into HTML tables and select tags, in addition to utilities useful for using SVG.

2.6.2 Launching web*Mathematica* Automatically

It is common for a web application, such as web*Mathematica,* to be launched automatically whenever the server machine starts. This section will review the ways that this can be done. Of course this is typically done by the system administrator; the information here is something that can be used to adapt to the conventions of your own system. The instructions are different for Unix and Windows. These instructions will focus on launching Tomcat; if you use some other servlet container, you will need to consult its documentation.

Unix

Tomcat can be launched automatically by adding it to a system startup script. The description here is typical of a Linux RedHat system. It is a good idea to first study the instructions for setting up Tomcat on Unix.

©1988-2006 Wolfram Research, Inc. All rights reserved.

A common way to launch is to add a script to the initialization directory `init.d` and making a link to the appropriate startup directory (for example `rc3.d`). The script will make definitions for starting and stopping Tomcat, making use of the `tomcat` user that was created. A sample script is shown below.

```
#!/bin/sh
#
# tomcat        This shell script takes care of starting and stopping
#               tomcat.
# description: tomcat is a servlet/JSP engine, which can be used
#              standalone or in conjunction with Apache


# Source function library.
. /etc/rc.d/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

RETVAL=0

export JAVA_HOME=/usr/local/java
export CATALINA_HOME=/usr/local/tomcat
export PATH=$PATH:/usr/local/bin:$JAVA_HOME/bin

# See how we were called.
case "$1" in
  start)
        # Start daemons.
        echo -n "Starting tomcat: "
        cd ~tomcat
        su tomcat -c '$CATALINA_HOME/bin/catalina.sh start'
        RETVAL=$?
        [ $RETVAL -eq 0 ] && touch /var/lock/subsys/tomcat
        echo
        ;;
  stop)
        # Stop daemons.
        echo -n "Shutting down tomcat: "
        cd ~tomcat
        su tomcat -c '$CATALINA_HOME/bin/catalina.sh stop'
        RETVAL=$?
        [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/tomcat
        echo
        ;;
  restart)
        $0 stop
        $0 start
        RETVAL=$?
        ;;
  *)
        echo "Usage: tomcat {start|stop|restart}"
        exit 1
esac

exit $RETVAL
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

In addition, if you are using a virtual X server, such as Xvnc, you will need to launch this at system initialization time. A sample script for launching Xvnc is shown below.

```
!/bin/sh
#
# description: VNC instance for webMathematica

# Source function library.
. /etc/rc.d/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

RETVAL=0

# See how we were called.
case "$1" in
  start)
        # Start daemons.
        echo -n "Starting VNC: "
        cd ~tomcat
        su tomcat -c '/usr/bin/vncserver -geometry 800x600 -depth 24
    :1 >/dev/null 2>/dev/null'
        RETVAL=$?
        [ $RETVAL -eq 0 ] && touch /var/lock/subsys/vnc
        echo
        ;;
  stop)
        # Stop daemons.
        echo -n "Shutting down VNC: "
        cd ~tomcat
        su tomcat -c '/usr/bin/vncserver -kill :1 >/dev/null
    2>/dev/null'
        RETVAL=$?
        [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/vnc
        echo
        ;;
  restart)
        $0 stop
        $0 start
        RETVAL=$?
        ;;
  *)
        echo "Usage: vnc {start|stop|restart}"
        exit 1
esac

exit $RETVAL
```

Windows

The way to run programs automatically on Windows is to install them as a service. This can only be done for Windows NT/2000/XP.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Installing Tomcat as a service on Windows is quite easy, since the installer will give you the option of installing it as a service. If you did not choose this as an option when you first installed Tomcat, you can rerun the installer.

You can get extra information about the services you have installed by going to the `Administrative Tools` icon of the `Control Panel`. Here you will see a `Services` icon which you can open. The `Services` dialog that opens will show the services that have been installed on your machine. You should see one for Tomcat. The dialog will tell you if the service has started or not and will tell you how it is launched. Typically the installer makes it an `Automatic` startup type which causes it to launch when the computer is booted. Clicking the Tomcat entry and selecting `Properties` will give you a dialog, that allows you to configure the service.

Running Tomcat as a service works well if the JDK is 1.4. The JDK is found by searching for java.exe on the Windows path, so you need to make sure that the JDK you installed earlier is found.

2.6.3 Web Server Connections

In some configurations, the main accessible server is a regular web server such as the Apache HTTP server, the Microsoft Internet Information Server, or the Netscape Enterprise Server. It is possible to use such a server in conjunction with a servlet container. This type of configuration is more complicated, but can take advantage of many additional features, for example, authentication and URL rewriting, and is often suitable for an existing web infrastructure. This section gives a brief description of configuration for some typical arrangements. It is probably useful to configure your servlet container to work in a stand-alone mode before starting any of this work.

If you wish to use web*Mathematica* in conjunction with a separate web server, you will need to make sure that all requests to web*Mathematica* are forwarded to the web*Mathematica* web application. This will make sure that requests for applet archives, HTML pages, and images are properly processed. If the server is only set to allow access to servlets, then these other resources will not be accessible.

If you just wish to test web*Mathematica* running directly through a servlet container such as Tomcat, you may skip this section.

Apache and Tomcat

There are a number of ways for the Apache web server to communicate with Tomcat. One convenient way is to use an HTTP forwarding mechanism to send requests from the Apache web server to Tomcat. This can be arranged with the `ProxyPass` configuration directive. A sample configuration, that could be added to the Apache configuration file (typically called `httpd.conf`), is shown below.

```
<IfModule mod_proxy.c>
  <Location /webMathematica/KernelMonitor>
    Order allow,deny
    deny from all
  </Location>

  ProxyPass /webMathematica http://tomcatserver:8080/webMathematica
  ProxyPassReverse /webMathematica
    http://tomcatserver:8080/webMathematica
</IfModule>
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

The ProxyPass directive provides a mapping from a path to an external URL, and the ProxyPassReverse directive causes the responses to be modified so that the proxy is transparent. Any access to /webMathematica will be sent to port 8080 on the machine tomcatserver. The configuration also denies access to the kernel monitor via the proxy. The features of the kernel monitor are described in a later section.

For the ProxyPass directive to be effective, the modules mod_proxy and mod_proxy_http must be loaded, this can be done with the following configuration information in the Apache configuration file.

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
```

For more information on using Apache and Tomcat, you should consult the web sites for Apache, http://httpd.apache.org, and Tomcat, http://jakarta.apache.org. Note that you can run Apache on both Windows and Unix machines, as well as Mac OS X machines. Another useful web site is http://www.apachetomcat.com/home.

Microsoft Servers and Tomcat

If you wish to deploy Microsoft PWS or IIS as your web server, it is possible to use them with Tomcat as a servlet container.

Tomcat comes with instructions for configuring it as a servlet container for IIS and PWS. The configuration has to be done manually, and, although straightforward, it has a number of steps that must be followed carefully and involves editing the Windows registry by hand. Because of these manual steps, it is perhaps easier to use JRun with these servers.

You need to make sure that you edit the workers.properties file, found in the Tomcat conf directory, to give values to workers.tomcat_home and workers.java_home. The required settings will be clear from the description in the file.

Apache/Microsoft Servers and JRun

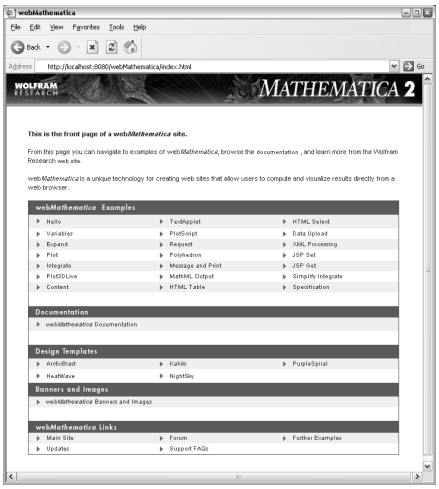
JRun provides a Connector Wizard, which is accessible through the JRun Management Console. This can be used to forward requests to web*Mathematica*.

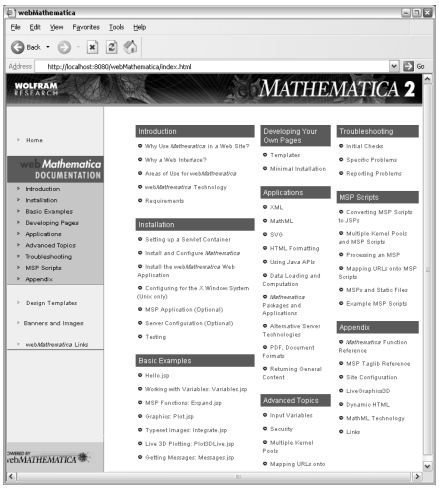
2.7 Testing

You should now be able to restart your server and connect to web*Mathematica*. Note that you may have to modify the URLs shown in this document to connect to your servlet container; for example, the URL may require a different port number than 8080, which is chosen as the default port for direct access to Apache Tomcat. The URL that you use to test your servlet container will show the correct URL to use for web*Mathematica*. In addition, you should note that the URL is case sensitive, so make sure to type in capitals as they appear.

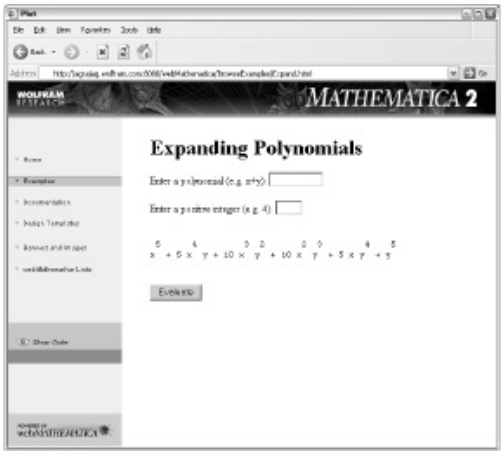
You may first want to connect to the web*Mathematica* front page via http://localhost:8080/webMathematica; this should look similar to the picture below. It contains links to examples, documentation, templates, images, and external references.

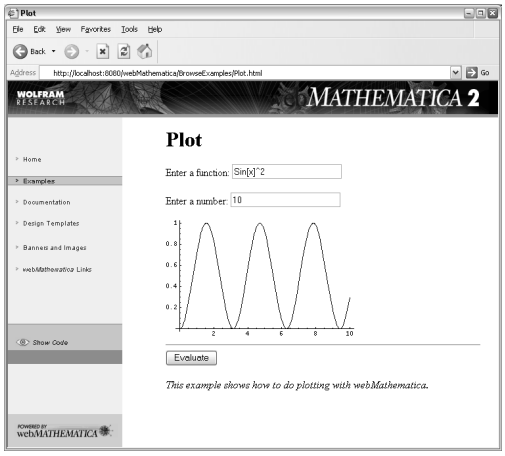
©1988-2006 Wolfram Research, Inc. All rights reserved.





After this, it may be good to try some of the active examples, such as `Expand`, which can be reached from a link on the front page and from a URL such as http://localhost:8080/webMathematica/BrowseExamples/Expand.html . Enter parameters into the input fields and hit `Evaluate`; it should look similar to that shown below.





It may also be a good idea to test a graphics example such as http://localhost:8080/webMathematica/Examples/-Plot.jsp. If you are running on Unix, you will need to configure your X server specially (as described here) to generate graphics and use other features of the *Mathematica* front end. It is a good idea to test that this works correctly.

If you have problems and cannot connect, go to the section on Troubleshooting.

©1988-2006 Wolfram Research, Inc. All rights reserved.

3. Basic Examples

This chapter covers a number of initial examples of web*Mathematica*. Many of these can be copied and used as the basis for your own work.

These examples are a form of JavaServer Pages (JSPs) that use a special library of tags that work with *Mathematica*. JSPs support the embedding of Java into HTML, and are frequently used with Java Servlets to develop large dynamic web sites. The library of tags is called the MSP taglib and will work on any compliant servlet engine. One advantage of the use of a tag library is that it can completely hide any use of the Java programming language; this is the case with the MSP taglib.

This chapter considers the basics of web*Mathematica* scripts. It requires some knowledge of HTML, including `form` and `input` elements. A reference to HTML is included at the end of this document. If you have no understanding of `form` elements, it will be hard to write interactive examples for web*Mathematica*.

The description given here will work through a collection of sample JSPs, each of which will demonstrate some detail or feature. The sources for all these examples are included in the web*Mathematica* web application in the directory `Examples` (the full path in Tomcat would be `webapps/webMathematica/Examples`). If you followed the installation steps when you installed your server, you should be able to see these examples running live in your server. Please note that these examples are designed to be simple examples of how to program with web*Mathematica* technology and have not been created for pleasing visual appearance.

These examples can be reached from the web*Mathematica* home page, which you should be able to reach via http://localhost:8080/webMathematica. (You may have some other URL for accessing your server.) The home page shows examples wrapped up in a template that adds more design around the pages to give them a better visual appearance. To study the details of how to program for web*Mathematica*, this extra design may be a distraction and it is also possible to reach the examples without using the template.

When you have finished this chapter, you may wish to look at the next chapter: Developing Your Own Pages. This gives some ideas for starting to develop your own site.

3.1 Hello.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/Hello.jsp. (You may have some other URL for accessing your server.)

This example evaluates the `Date[]` function of *Mathematica*. The result changes each time the page is accessed, demonstrating that this really is a dynamic process. The source for this page is in `webMathematica/Examples/Hello.jsp`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %> standard jsp
                                                        headers

<html>                                          standard html tags
    <head>
        <title>Hello World (The MSP Taglib)</title>
    </head>
    <body>
        <h1>Hello World (The MSP Taglib)</h1>
        <msp:allocateKernel>                    allocate a Mathematica kernel
            <h4>Date[]</h4>
        <msp:evaluate>
        Date::usage                             evaluated by Mathematica
        </msp:evaluate>

        <p>Its current value is:</p>
        <msp:evaluate>
             Date[]                             evaluated by Mathematica
        </msp:evaluate>
        </msp:allocateKernel>                   free the Mathematica kernel
    </body>
</html>
```

This page uses standard HTML tags as well as special web*Mathematica* tags; these have the form `<msp:tag>`. The web*Mathematica* tags are executed from the top of the page to the bottom. The `<msp:allocate>` tag causes a *Mathematica* kernel to be allocated to use for computations. The contents of the `<msp:evaluate>` tags are sent to *Mathematica* for computation with the result inserted into the final page. The `</msp:allocate>` tag frees the *Mathematica* kernel to be used for another computation.

3.2 Working with Variables: Variables.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this MSP via http://localhost:8080/webMathematica/Examples/Variables.jsp. (You may have some other URL for accessing your server.) It demonstrates how variables are connected to input values. The source for this page is in `web Mathematica/Examples/Variables.jsp`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<html>
<head>
<title>Assigning Variables</title>
</head>
<body bgcolor="#ffffff" >
<h1>Assigning Variables</h1>

<form action="Variables.jsp" method="post">      html form
<msp:allocateKernel>
Enter something:
<br>
<input type="text" name="tmp" size="10" />       field for user input
<br>
<br>
<msp:allocateKernel>
<msp:evaluate>$$tmp</msp:evaluate>               evaluated by Mathematica

</msp:allocateKernel>
<br>
<input type="submit" name="submitButton"
                        value="Evaluate" />      button for activating form
</msp:allocateKernel>
</form>
</body>
</html>
```

This page is more elaborate because it contains `form` and `input` elements. These are important ways for allowing interaction from the client.

A `form` element is a block of HTML that may contain `input` elements. A `form` may be activated with an `input` of type `submit`; this sends the name and value associated with each `input` tag to the server. Here, the opening tag of the `form` element contains two attributes. The `action` attribute refers to a URL that is accessed when the `form` is activated. In this case, it is a relative URL that refers to the original Variables script. The `method` attribute tells the browser what HTTP method to use, in this case, a `post` method. (It is very common to use `post` methods.)

This example has two `input` tags: the first allows the user of the page to enter text, and the second specifies a button that, when pressed, will submit the `form`. When the `form` is submitted, it will send information from `input` elements to the URL specified by the `action` attribute (in this case, the same JSP). Text entered into the `input` tag, which uses the name `tmp`, will be assigned to the input variable `$$tmp`.

The first time the page is accessed there is no value for `$$tmp`. When a value is entered in the text field and the `Evaluate` button pressed, `$$tmp` gets a value which is displayed. Note that the value is a *Mathematica* string—if you try and enter a computation such as `"5+7"`, no computation is actually done. If you want the input to be interpreted and evaluated by *Mathematica*, you need to use one of the MSP functions that are described later in this chapter.

Note that the `$$` prefix is used to label input variables; these are variables that are sent with the HTTP request. The use of variables is discussed further in Tips and Tricks: Variables.

©1988-2006 Wolfram Research, Inc. All rights reserved.

3.3 MSP Functions: Expand.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/Expand.jsp. (You may have some other URL for accessing your server.)

When the `submit` button is pressed, the polynomial is raised to the power and expanded, and an HTML page that contains the result is returned. The source for this page is in `webMathematica/Examples/Expand.jsp`. A section that shows the `form` tag is shown below.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<html>
<head>
<title>Expanding Polynomials</title>
</head>
<body>
<h1>Expanding Polynomials</h1>
<form action="Expand.jsp" method="post">
Enter a polynomial (e.g. x+y):
 <input type="text" name="expr" size="10">
Enter a positive integer (e.g. 4):
 <input type="text" name="num" size="3">
<br>
<msp:allocateKernel>
<msp:evaluate>
    MSPBlock[{$$expr,$$num},
        Expand[$$expr^$$num]]        secure computation with input variables
</msp:evaluate>
</msp:allocateKernel>
<input type="submit" name="submitButton" value="Evaluate">
</form>
</body>
</html>
```

This page contains `form` and `input` tags as described in the previous example. Additionally, the `msp:evaluate` tag refers to the MSP function `MSPBlock`.

When the `form` is submitted, the server connects to a *Mathematica* kernel, in which two symbols, `$$expr` and `$$num`, are assigned to the text from the two `input` elements. If no text is entered, the symbols will not have any definition.

Mathematica now evaluates the contents of the `msp:evaluate` tag. The `MSPBlock` command is a programming construct, which here inspects two input variables, `$$expr` and `$$num`. If either of these has no value, `MSP Block` returns a null string, which is why the first time you access the page, you do not see a result. The values of both variables are then interpreted by *Mathematica*. If successful, the results of interpretation are substituted into the second argument or body of `MSPBlock`. In this example all instances of `$$expr` are substituted with the parsed value of `$$expr`, and the same is done for `$$num`. The result is then evaluated, formatted, and placed in the HTML page, which is returned to the client.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Interpretation of the variables by *Mathematica* can fail in two ways: the input might not be valid *Mathematica* input (for example, `f[}`), or it might be dangerous input (such as `ReadList["/etc/passwd"]`). In both cases, the inputs are rejected and an error message generated. This demonstrates some of the security features of the system, which the Security section documents in detail. The use of variables is discussed further in Tips and Tricks: Variables.

The formatting of the result of an `msp:evaluate` tag is discussed in more details in Advanced Topics: Evaluation Formatting.

3.4 Graphics: Plot.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/Plot.jsp. (You may have some other URL for accessing your server.)

This example generates a plot. The source for this page is in `webMathematica/Examples/Plot.jsp`. A section that shows the `form` tag is shown below.

```
<form action="Plot.jsp" method="post">

<msp:allocateKernel>
Enter a function:
<input type="text" name="fun" size="24" value =
    "<msp:evaluate>MSPValue[ $$fun,  "Sin[x]^2"]
        </msp:evaluate>"        remember input settings from one call to the next
/>
Enter a number:
<input type="text" name="x1" size="24" value =
    "<msp:evaluate>MSPValue[ $$x1,  "10"]
        </msp:evaluate>"
/>
<br>
<msp:evaluate>
    MSPBlock[ {$$fun, $$x1},
        MSPShow[ Plot[$$fun, {x,0,$$x1}]]]
</msp:evaluate>
<hr>
</msp:allocateKernel>

<input type="submit" name="btnSubmit" value="Evaluate">

</form>
```

The example shows the use of the MSP functions `MSPBlock` and `MSPShow`. `MSPBlock` is a programming construct introduced in the previous section. `MSPShow` takes the *Mathematica* graphics object from the `Plot` command and generates a GIF image, which is stored on the server, returning an `<img>` tag. A further discussion on formatting mathematics and graphics is given in the section on Displaying Mathematics and Graphics. Note how the page uses `MSPValue` to keep the user input each time the page is used.

©1988-2006 Wolfram Research, Inc. All rights reserved.

3.5 Typeset Images: Integrate.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/Plot.jsp. (You may have some other URL for accessing your server.)

This example allows the user to enter a function to be integrated. The result is then formatted by the typesetting system and saved as an image. The source for this page is in `webMathematica/Examples/Integrate.jsp`. A section that shows the `form` tag is shown below.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<html>
<head>
<title>Integration</title>
</head>
<body>
<h1>Integrate a function</h1>
<form action="Integrate.jsp" method="post">Input:
<br>
<msp:allocateKernel>
<msp:evaluate>
    integrand = Null;                page variable to hold the integrand
    If[ MSPValueQ[ $$expr],
        integrand =                  interpret input with secure conversion
            MSPToExpression[ $$expr]] ;
</msp:evaluate>

<input type="text" name="expr"  size="24"
          value="<msp:evaluate> MSPValue[ $$expr,
    "Sin[x]^2"]</msp:evaluate>" />
<br/>
<br/>
<msp:evaluate>                       carry out the integration
    If[ integrand =!= Null,
        MSPFormat[
            Integrate[integrand,x],StandardForm]]
</msp:evaluate>

</msp:allocateKernel>
<br/>
<input type="submit" name="btnSubmit" value="Evaluate">
</form>
</body>
</html>
```

In this example, an `msp:evaluate` tag integrates an expression and uses `MSPFormat` to format the result with `StandardForm`. This generates an image and returns a reference to the image. For this to work, it is necessary to use the *Mathematica* front end.

©1988-2006 Wolfram Research, Inc. All rights reserved.

The example also demonstrates the use of page variables with MSPToExpression. This is an alternative to using MSPBlock suitable in certain constructions, for example, when the input will be used in a number of computations. The page variable integrand is initialized to Null and later, if its value has been modified, the integration is carried out. It is assigned to the interpreted value of $$expr only if this input variable actually has a value. Note that if an error, such as a security error, is encountered in interpreting $$expr, an exception will be thrown and integrand will remain assigned to Null.

Note that MSPToExpression applies a security check to the input variable. You should be aware that input variables are a major source of danger and always use the secure conversion functions MSPBlock and MSPToExpression. In particular, you should never use ToExpression on an input variable. The Security section documents the security system in more detail.

It is also possible to return the result using MathML; this is described in greater detail in the section on MathML. A further discussion on formatting mathematics and graphics is given in the section on Displaying Mathematics and Graphics.

An interesting point about the first msp:evaluate tag is that it contains two *Mathematica* commands. To use two commands in the same tag they can be separated with a semicolon ';'. In addition, the last command is also followed by a semicolon, this makes sure that no output from the tag is inserted into the output page. More information on adding code into web *Mathematica* pages is given in the section Tips and Tricks: Coding in Pages.

3.6 Live 3D Plotting: Plot3DLive.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/Plot3DLive.jsp. (You may have some other URL for accessing your server.) It allows the user to enter a function to be plotted by the LiveGraphics3D applet. The source for this page is in webMathematica/Examples/Plot3DLive.jsp.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<html>
<head>
<title>Live 3D Plotting</title>
</head>

<body text="#171717" bgcolor = "#ffffff">
<msp:allocateKernel>

<msp:evaluate>
 $ImageBackground = "#ffffff";
 $ImageSize = {300,300};
</msp:evaluate>

<h1>Live 3D Plotting</h1>
<form action="Plot3DLive.jsp" method="post">
  Plot3D of
  <input type="text" name="fun"  size="22" value =
     "<msp:evaluate> MSPValue[ $$fun,  "Sin[x y]^2"] </msp:evaluate>">
  <br>
  x from:
  <input type="text" name="x0"  size="10" value =
    "<msp:evaluate>MSPValue[ $$x0,  "-2"] </msp:evaluate>">
  to:
  <input type="text" name="x1"  size="10" value =
    "<msp:evaluate>MSPValue[ $$x1,  "2"] </msp:evaluate>">
  <br>
  y from:
  <input type="text" name="y0"  size="10" value =
    "<msp:evaluate>MSPValue[ $$y0,  "-2"] </msp:evaluate>">
  to:
  <input type="text" name="y1"  size="10" value =
    "<msp:evaluate>MSPValue[ $$y1,  "2"] </msp:evaluate>">
  <br>
  Number of points to plot
  <input type="text" name="pts"  size="5" value =
    "<msp:evaluate>MSPValue[ $$pts,  "20"] </msp:evaluate>">
  <br>
  <msp:evaluate>
    MSPBlock[ {$$fun, $$x0, $$x1, $$y0, $$y1, $$pts},
            MSPLive3D[
                Plot3D[$$fun,
                {x,$$x0,$$x1}, {y,$$y0,$$y1}, PlotPoints -> $$pts]]]
  </msp:evaluate>
  <br><hr>
  <input type="submit" name="btnSubmit" value="Evaluate">
</form>
</msp:allocateKernel>
</body>
</html>
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

This example uses a number of evaluations to set up parameters. The last evaluation takes the values of these parameters and uses them in a call to `Plot3D`. The result of this goes to `MSPLive3D`, which calls the `Live Graphics3D` applet. This gives a real-time rotation of the three-dimensional graphics object.

3.7 Getting Messages: Messages.jsp

This example demonstrates how messages and print output generated by *Mathematica* can be returned in the web page. If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/Messages.jsp. (You may have some other URL for accessing your server.) The source for this page is in `webMathematica/Examples/Messages.jsp`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<html>
<head>
<title>Messages and Print Output</title>
</head>
<body text="#171717" bgcolor = "#ffffff">
<h1>Messages and Print Output</h1>

<p>These are some evaluations that will cause messages to be
generated by Mathematica.</p>

<msp:allocateKernel>
<h4>Input is 1/0</h4>

<msp:evaluate>
    Random[]/0
</msp:evaluate>

<h4>Input is Sin[x,1]</h4>

<msp:evaluate>
    Sin[x,1]
</msp:evaluate>

<h4>Input is Print[ "The result is ", x^2]</h4>

<msp:evaluate>
    Print[ "The result is ", x^2]
</msp:evaluate>

<hr><p>The messages were:</p>

<msp:evaluate>
    ColumnForm[ MSPGetMessages[]]        messages displayed here
</msp:evaluate>

<hr><p>The print output was:</p>

<msp:evaluate>
    ColumnForm[ MSPGetPrintOutput[]]     print output displayed here
</msp:evaluate>

<hr>
</msp:allocateKernel>
</body>
</html>
```

The contents are very simple; there are two evaluations that cause messages to be generated. These are followed by uses of `MSPGetMessages` and `MSPGetPrintOutput`, both of which are formatted by `ColumnForm`. The messages that were generated are displayed in the resulting page.

©1988-2006 Wolfram Research, Inc. All rights reserved.

3.8 Returning General Content: Content.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/Content.jsp. (You may have some other URL for accessing your server.)

All of the examples up to this point return HTML to the browser, but the web can work with general content involving many different formats. `MSPReturn` is provided to allow an MSP to return arbitrary content. Here is an example that demonstrates how different formats can be returned. The source is in `webMathematica/Examples/Content.jsp` and `webMathematica/WEB-INF/Applications/ExampleUtilities/Content.m`.

First, here is the source.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<html>
<head>
<title>Live 3D Plotting</title>
</head>

<body text="#171717" bgcolor = "#ffffff">
<html>
<head>
<title>General Content</title>
</head>
<body bgcolor="#ffffff">
<h1>General Content</h1>

<form action="Content.jsp" method="post">
This example takes a format type and converts a notebook into
this format type.  It returns the converted notebook.  </p>

<msp:allocateKernel>
<msp:evaluate>
    If[ MSPValueQ[ $$button],
        Get[ "ExampleUtilities`Content`"];
        MSPReturn @@ GeneralContent[ $$button]]
</msp:evaluate>

</msp:allocateKernel>

<p>
Please select a format:
</p>
<input type="submit" name="button" value="Notebook">
<input type="submit" name="button" value="PostScript">
<input type="submit" name="button" value="GIF">
</form>
</body>
</html>
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

Here is the *Mathematica* source.

```
MakeNotebook[] :=
    Developer`UseFrontEnd[
        Module[ {nb, nbobj},
            nb = NotebookCreate[] ;
            NotebookWrite[ nb, Cell[ "A Dynamically Created
    Notebook", "Title"]] ;
            NotebookWrite[ nb,
                Cell[ "Converted to " <> $$button, "Subtitle"]] ;
            NotebookWrite[ nb, Cell[ "The date is " <> ToString[
    Date[]], "Text"]] ;
            nbobj = NotebookGet[ nb] ;
            NotebookClose[ nb] ;
            nbobj]]


GeneralContent[ fmt_] :=
    Module[ {nbobj},
        nbobj = MakeNotebook[] ;
        Developer`UseFrontEnd[
            Switch[
                fmt,
                    "Notebook",
                        {ToString[ nbobj, InputForm],
                                "application/mathematica"},
                    "PostScript",
                        {DisplayString[ nbobj, "EPS"],
                                "application/eps"},
                    "GIF",
                        {DisplayString[ nbobj, "GIF"],
                                "image/gif"},
                    _, "Unknown format"]
                ]]
```

In this example, one evaluation tests the variable `$$button`. If it has a value from activating one of the buttons in the `form`, this is used to specify a return format type and passed to a function, `GeneralContent`. The *Mathematica* code for this function is placed into a separate package to be loaded when the variable is set. `GeneralContent` calls a function that creates a very simple notebook, `MakeNotebook`. `MakeNotebook` generates a notebook using the *Mathematica* notebook API and the function `Developer`UseFrontEnd`. In a real life situation a more interesting notebook would probably be generated. `MSPReturn` returns the representation of the notebook to the server with the content type. This is then returned to the browser, which, if suitably configured, will deploy the necessary helper application.

In a more advanced example, the dynamically generated notebook would probably use information sent with the request from the client.

If you wish to return special content and also set a filename to be used with this content, then you may wish to use the three-argument form of `MSPReturn`. This is described in the MSP Function Reference section.

Another way to set the content returned from an MSP script is to use `MSPPageOptions`. The topic of returning general content is discussed later.

©1988-2006 Wolfram Research, Inc. All rights reserved.

3.9 Applets: TextApplet.jsp

This example demonstrates how to call on the services of a *Mathematica*-powered web site from an applet. This shows a combination of client and server programming. The section involves some programming in Java.

If you installed web*Mathematica* as described above, you should be able to connect to this MSP via http://localhost:8080/webMathematica/Examples/TextApplet.jsp. (You may have some other URL for accessing your server.) The source for this page is in `webMathematica/Examples/TextApplet.jsp` and `webMathematica/WEB-INF/src/ExampleApplets/TextApplet.java`.

First, here is the JSP source.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<html>
<head>
<title>Live 3D Plotting</title>
</head>

<body text="#171717" bgcolor = "#ffffff">
<html>
<title>Applet Test</title>
<body bgcolor="#ffffff">
<p>
Here is an applet that gets a result from Mathematica:
<br>

<msp:allocateKernel>
<msp:evaluate>
    If[ MSPValueQ[ $$Compute],
        MSPReturn[ "Date[] returns " <> ToString[ Date[]],
    "text/plain"]]
</msp:evaluate>
<applet
    code="TextApplet"
    archive =
        "<msp:evaluate> $WebApplication <>
    "/Resources/applets/MSPExamples.jar"</msp:evaluate>"
    width="400"
    height="30" >
    <param name="ArgumentURL" value="TextApplet.jsp?Compute=True">
</applet>
</msp:allocateKernel>

<p>
Hitting refresh will cause the page to update.
</body>
</html>
```

Here is the source for the applet `TextApplet.java`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
import java.applet.Applet;
import java.awt.*;
import java.net.*;
import java.io.*;
public class TextApplet extends Applet
{
  public void paint(Graphics g)
  {
    super.paint(g);
    try {
      URL url=new URL(getDocumentBase(),getParameter("ArgumentURL"));
      InputStream in=url.openStream();
      ByteArrayOutputStream out=new ByteArrayOutputStream();
      byte[] b=new byte[1024]; int len;
      while((len=in.read(b,0,1024)) != -1) {
        out.write(b,0,len);
      }
      b=out.toByteArray();
      g.drawBytes(b,0,b.length-1,20,20);
    }
    catch (Exception e) {
      System.out.println("Error "+e);
    }
  }
}
```

This is a very simple applet; the paint method opens a connection to a URL, the name of which is formed from the document that loaded the applet, and the value of the parameter `ArgumentURL`, which is passed in from a `param` tag. This causes the TextApplet JSP to be called and return a computation of the date.

3.10 JavaScript: PlotScript.jsp

This example demonstrates how to integrate a *Mathematica*-powered web site with JavaScript. It also demonstrates both client and server programming. The section involves some programming in JavaScript.

Note that JavaScript and Java are different languages. JavaScript is a scripting language that is useful for manipulating documents and other features of browsers. Java is a general purpose programming language that can be used in an HTML document via an applet. The two languages complement each other: JavaScript is useful for manipulating the browser and documents that are open in the browser, while Java has a more sophisticated collection of functions and can draw into the browser window. It is possible for JavaScript and Java to work together.

If you installed web*Mathematica* as described above, you should be able to connect to this MSP via http://localhost:8080/webMathematica/Examples/PlotScript.jsp. (You may have some other URL for accessing your server.) The source for this page is in `webMathematica/Examples/PlotScript.jsp` and `webMathematica/Examples/PlotScript1.jsp`.

First, here is the source for `PlotScript.jsp`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<html>
<head>
<title>Live 3D Plotting</title>
</head>

<body text="#171717" bgcolor = "#ffffff">
<html>
  <head>
    <SCRIPT>
      <!--- Hide script from old browsers
        function plot(f) {
          win = window.open( "PlotScript1?fun=" +
                escape(f.fun.value) + "&x1=" + escape(f.x1.value),
                "plot","toolbar=none");
          win.resizeTo( 450, 350);
         }
        // end hiding from old browsers -->
    </SCRIPT>
    <title>Plot</title>
  </head>
  <body>
    <h1>Plot</h1>
    <form action="Plot" method="post">
     <msp:allocateKernel>
      Enter a function:
      <input type="text" name="fun"  size="24" value =
         "<msp:evaluate> MSPValue[ $$fun,  "Sin[x]^2"]
    </msp:evaluate>">
      Enter a number:
      <input type="text" name="x1"  size="24" value =
         "<msp:evaluate> MSPValue[ $$x1,  "10"] </msp:evaluate>">
      <hr>
      <input type="button" value="Plot" onClick="plot(this.form)">
     </msp:allocateKernel>
   </form>
  </body>
</html>
```

Second, here is the source for `PlotScript1.jsp`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<html>
<head>
<title>Function Plot</title>
</head>
<body bgcolor="#ffffff">
<msp:allocateKernel>
<msp:evaluate>
    MSPBlock[ {$$fun, $$x1},
        MSPShow[ Plot[$$fun, {x,0,$$x1},ImageSize->400]]]
</msp:evaluate>
</msp:allocateKernel>
<form action="Plot" method="post">
<input type="button" value="Close" onClick="window.close()">
</form>
</body>
</html>
```

This is a simple example given to demonstrate how a JSP can work with JavaScript. The initial page, `PlotScript.jsp`, puts up a page with a `form` of two text `input` elements and one `submit` button. When the button is clicked, it opens a new window that contains the output of `PlotScript1.jsp`.

3.11 Setting Variables: SetBasic.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/SetBasic.jsp. (You may have some other URL for accessing your server.)

This example passes values computed in a JSP into *Mathematica* where they are used for computation by *Mathematica*. This example uses the Java programming language which makes it different from most web*Mathematica* examples which do not require any Java programming. The source for this page is in `webMathematica/Examples/SetBasic.jsp` and is shown below.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<html>
<head>
<title>Set Example</title>
</head>
<body bgcolor="#ffffff" >

<h1>Set Example</h1>

<p>
This jsp uses the MSP Taglib to demonstrate passing
Java values to Mathematica.
</p>

<%
  int num = 10;
  Object obj = new Object();
%>

<msp:allocateKernel>

<msp:set name="var1" intValue="<%= num %>" />
<msp:set name="var2" objectValue="String from java" />
<msp:set name="var3" objectValue="<%= obj %>" />

<msp:evaluate>
  Nest[ f, x, var1]
</msp:evaluate>

<msp:evaluate>
  Characters[ var2]
</msp:evaluate>

<msp:evaluate>
  var3@hashCode[]
</msp:evaluate>

</msp:allocateKernel>

</body>
</html>
```

In this example, a variable `num`, which is an int, `str`, which is a String, and `obj`, which is an Object, are created in the JSP. These are then passed to *Mathematica* using the `msp:set` tag. This tag takes two attributes, the `name` attribute gives the name that the variable will be given in *Mathematica*, while the `value` attribute refers to the value. If the variable is of a primitive type, such as int, char, or double, then it needs to use the appropriate value attribute, such as `intValue`, `charValue`, or `doubleValue`. Notice how `msp:set` sends a Java int as a *Mathematica* integer and a Java String as a *Mathematica* string. The Java Object is sent as a *Mathematica* object reference. The rules that govern how types are sent from Java to *Mathematica* are exactly those that *J/Link* uses.

©1988-2006 Wolfram Research, Inc. All rights reserved.

3.12 Getting Variables: GetBasic.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/GetBasic.jsp. (You may have some other URL for accessing your server.)

This example passes values computed in a JSP into *Mathematica* where they are used for computation by *Mathematica*. The source for this page is in `webMathematica/Examples/GetBasic.jsp`, and a selection is shown below.

```
<msp:allocateKernel>

<msp:evaluate>
    num = Random[Integer, {1,10}];
    list = Table[ Random[], {num}] ;
    sum = Apply[ Plus, list]/num;
</msp:evaluate>

<msp:get name="dArray" type="double[]" value="list" />
<msp:get name="dValue" type="Double" value="sum" />

<p>
The average is <%= dValue %>.
</p>

<table border>
<% for ( int i = 0; i < dArray.length; i++) { %>
<tr><td><%= dArray[i] %></td></tr>
<% } %>
</table>

</msp:allocateKernel>
```

In this example *Mathematica* generates a list of random numbers, and computes the mean. The JSP obtains these values using the `msp:get` tag. This has a `name` attribute, which governs the name of the variable in Java, a `type` attribute, which governs the type that will be used in Java, and a `value` attribute, which gives the *Mathematica* expression that must be evaluated. Only reference types such as Object, String, or arrays can be sent with this tag. In addition the type given in Java must be compatible with the type that *Mathematica* has sent; if this is not the case, then an exception will be thrown. The rules for transmission from *Mathematica* are that the normal *J/Link* type conversions will be applied, but if none of these applies, then the object will be converted into an object of type com.wolfram.jlink.Expr, a class that is provided by *J/Link* to represent general *Mathematica* expressions.

This particular example requires programming in both Java and *Mathematica*. It shows how easy it would be to incorporate web*Mathematica* into an existing JSP framework.

©1988-2006 Wolfram Research, Inc. All rights reserved.

3.13 Interacting with MSPs: IncludeMSP.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/IncludeMSP.jsp. (You may have some other URL for accessing your server.)

This example demonstrates how a JSP can call an MSP script. MSP scripts are the original form of web*Mathematica* interaction that are now superseded by JSPs; they are described in Classic web*Mathematica* Technology. The source for this page is in `webMathematica/Examples/IncludeMSP.jsp`, and a section is shown below.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<html>
<head>
<title>MSP Include Example</title>
</head>
<body>

<h1>MSP Include Example</h1>

<p>
This JSP uses the MSP Taglib to demonstrate including
the results of an entire MSP.
</p>

<msp:includeClassicMSP name="Examples/PlotSingle"
    arguments="fun=Sin[x]&x1=10" />

</body>
</html>
```

In this example the `<msp:includeClassicMSP>` tag is used to include the result of the MSP script `Examples/PlotSingle.msp`. There are two attributes: `name`, which gives the MSP to be called, and `arguments`, which gives extra arguments for the MSP. This functionality is useful if you have existing MSPs and you wish to use them in a JSP. There is a closely related tag, `msp:forwardClassicMSP`, which forwards to an MSP.

©1988-2006 Wolfram Research, Inc. All rights reserved.

4. Developing Your Own Pages

Once you have installed and configured a web*Mathematica* server so that the examples run correctly and have studied the basics of writing material for web*Mathematica*, as described previously, you are ready to start developing your own material.

One way to start is to make your own area in the web*Mathematica* web application. You could make a directory here (for example `NewScripts)` and copy one of the samples (for example `Plot.jsp`) from the `Examples` directory. You could then access this script with the URL `http://localhost:8080/webMathematica/NewScripts/Plot.jsp`.

web*Mathematica Author* is an authoring system that aids in the creation and development of web*Mathematica* material. A preview version is shipped with web*Mathematica* 2.3. You can install and use it to help create new material for your web*Mathematica* site.

This might be a good time to revisit the web*Mathematica* index page found at http://localhost:8080/webMathematica/index.html, which provides a number of links that demonstrate features of web*Mathematica*. When you actually want to write your own material you may look at the Tips and Tricks described in this chapter. The chapter continues to describe other page development utilities that are part of web*Mathematica*.

4.1 Tips and Tricks

This section provides a summary of a few issues that will help you to get started writing your own pages. These are all described in more detail in later sections of the User Guide, but are collected together here in a brief description. Getting a good grasp of these points will help you to make progress in developing your site.

4.1.1 Variables

There are two types of variable that are important to understand when you are getting started with web*Mathematica*: input variables and page variables.

Input variables come with the HTTP request, for example from an input field in an HTML form. You can identify input variables in *Mathematica* code because they are labeled with a '$$' prefix. In the example below the `set ting` variable may be set by an input field. In *Mathematica* code it is called `$$setting`.

```
<input type="text" name="setting" />

<msp:evaluate>
    If[ MSPValueQ[ $$setting],
        ....
        ]
</msp:evaluate>
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

You should be aware that input variables are a potential security risk to your server and should always use the special functions, MSPBlock and MSPToExpression, for converting into *Mathematica* input. In particular, you should never use `ToExpression` on an input variable. An example of using MSPBlock is shown below.

```
<input type="text" name="fun" />

<msp:evaluate>
    MSPBlock[ {$$fun},
        Integrate[ $$fun, x]
        ]
</msp:evaluate>
```

Page variables are *Mathematica* variables that you use to hold intermediate values. They are called page variables since they are cleared when the page is finished. In the example below, the page variable `tmp` is used to hold the expression that was entered into the text input field (which is held in an input variable called `$$expr`). Note the use of the secure function, MSPToExpression, to convert the *Mathematica* expression from the input.

```
<input type="text" name="expr" />

<msp:evaluate>
    tmp = Null;
    tmp = MSPToExpression[ $$expr] ;
</msp:evaluate>

<p>
<msp:evaluate>
    If[ tmp =!= Null,
        ....
        ]
</msp:evaluate>
</p>
```

If you want your variable to persist from one page to another, you can declare it as a session variable. This and further details of variables are discussed in detail in Advanced Topics: Variables.

4.1.2 Coding in Pages

The purpose of web*Mathematica* is to use *Mathematica* for web computation; a key part of this is placing *Mathematica* code in your web pages. This is done with `<msp:evaluate>` tags, as follows.

```
<msp:evaluate
    Integrate[ 1/(1-x^3),x]
</msp:evaluate>
```

Note that the *Mathematica* code will evaluate in the typical way for *Mathematica* and the result of the computation will appear in the web page. You can use MSPFormat to change the way that the result is formatted; more information on formatting in web*Mathematica* is found in Advanced Topics: Evaluation Formatting. An example of

©1988-2006 Wolfram Research, Inc. All rights reserved.

MSPFormat is shown below; this formats the integral into TraditionalForm using a GIF image to display the result.

```
<msp:evaluate
    MSPFormat[ Integrate[ 1/(1-x^3),x], TraditionalForm]
</msp:evaluate>
```

If you do not wish to see the result in the web output, you can suppress it by using a semicolon ';'. In the following example an assignment is made to the variable x, but no output appears.

```
<msp:evaluate
    x = 109;
</msp:evaluate>
```

A final tip for working with code in web*Mathematica* pages is the separation of multiple computations in a single <msp:evaluate> tag by using a semicolon ';'. This is shown below.

```
<msp:evaluate
    x = 109;
    y = 44.5;
    {x+y}
</msp:evaluate>
```

More information on coding in web*Mathematica* pages is found in Appendix: MSP Taglib, which gives a detailed reference on the web*Mathematica* tags.

4.2 Templates

web*Mathematica* provides a number of templates and other utilities that can be used to incorporate more design into your web*Mathematica* material.

4.2.1 Browse Examples

The web*Mathematica* examples can be reached from the web*Mathematica* home page, which you should be able to reach via http://localhost:8080/webMathematica. (You may have some other URL for accessing your server.) The home page shows examples wrapped up in a template that adds more design around the pages to give them a better visual appearance. This template makes use of HTML frames and so it would be relatively easy to modify your own work to make use of it.

©1988-2006 Wolfram Research, Inc. All rights reserved.

4.2.2 Design Examples

As you develop your own material, you may wish to look at the design examples. These are a collection of samples that make use of colors, fonts, and images for a more professional appearance. You can access the design examples from the main index page, or with a URL such as http://localhost:8080/webMathematica/Examples/Design-Templates/PurpleSpiral.jsp. Each example has a link to the other examples.

4.2.3 Banners and Buttons

A collection of banners and buttons are available for use in your pages, which you can find with links http://localhost:8080/webMathematica/Resources/images/banners/ and http://localhost:8080/webMathematica/-Resources/images/buttons/. To use one of these images, such as the banner `webm-white.gif`, you can use an `img` tag such as the following.

```
<img src="/webMathematica/Resources/images/banners/webm-white.gif" />
```

The section on including static files has more information on how to include images.

Certain license options for web*Mathematica* require that you use an approved banner for your site. You may use one of these banner images in order to comply with this requirement.

4.3 Minimal Installation

When you have confirmed that your web*Mathematica* site is running correctly and you start to develop your own material, you may wish to strip out all of the documentation and examples to get a minimal installation. The minimum set of files for web*Mathematica* is shown below (the normal file layouts are described in Appendix: File Layout).

©1988-2006 Wolfram Research, Inc. All rights reserved.

4.3.1 Minimal File Layout

```
webMathematica
    WEB-INF
        web.xml
        Applications
            JLink
                JLink.m
                Kernel
                    (Mathematica code for J/Link)
            MSP
                MSP.m
                Security.m
                Kernel
                    init.m
        conf
            MSP.conf
            webMathematica.tld
        lib
            kernelpool.jar
            JLink.jar
            servlets.jar
            tags.jar
            webutils.jar
            SystemFiles
                Libraries
                    $SystemID
                        (JLink native library)
```

Note that you can remove all the *J/Link* native libraries from `webMathematica/WEB-INF/lib/System Files/Libraries` except for the library required for your system, which is located in a directory named by `$SystemID`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

5. Applications

This section shows how to use web*Mathematica* in a number of specific applications.

5.1 XML

XML is a general data format that is becoming increasingly important. Data that is formatted in XML can readily be used by applications that are able to process it. In this case the choice of an XML format means that you will save considerable development effort. In addition there are an increasing number of existing data formats that use XML. Some of the more important for mathematical and scientific purposes include XHTML (an XML compliant version of HTML), MathML (a way to store mathematical information), and SVG (a graphics format). A large list of XML applications is available at http://www.xml.org.

Mathematica contains a large number of features for working with XML, all of which are available in web*Mathematica*. XML can be very useful for web*Mathematica* with its support for specific XML applications and as a general format for data interchange. The use of MathML, SVG, and XHTML will be covered in their own sections. This section will give an overview of XML and the XML features of *Mathematica*. It will also give some examples of why this functionality is useful to web*Mathematica*.

5.1.1 Introduction to XML

This section will give a very brief introduction to XML. For more information, go to one of the many references such as those detailed at http://www.w3.org/XML/, for example, http://www.w3.org/XML/1999/XML-in-10-points.

A sample XML document is shown below.

```
<?xml version="1.0"?>
<library>
    <book>
        <title>A New Kind of Science</title>
        <author>Stephen Wolfram</author>
    </book>
    <book>
        <title>The Lord of the Rings</title>
        <author>J.R.R. Tolkien</author>
    </book>
</library>
```

In the example above we see a data format for a library. The library contains books and each book has a title and an author. This shows how XML is suitable for structured data. In addition we see how XML looks a little like HTML, except that the tags (words bracketed by '<' and '>') are not restricted to a fixed set since new tags can be introduced that are suitable for a particular application. Unlike HTML, the format of the XML is stricter with a valid XML document being required to follow rules that do not apply to HTML. This is demonstrated in the next section.

©1988-2006 Wolfram Research, Inc. All rights reserved.

XML Compliance

One issue with XML is that documents must be well-formed, following the rules of XML. Some basic examples of compliance are described in this section.

An XML document must include a header. For example, it must start with something like the following.

```
<?xml version="1.0"?>
```

Empty elements must either have an end tag, or the start tag must end with />. Thus, the following is legal.

```
<br/><hr/>
```

However, this is not legal.

```
<br><hr>
```

For nonempty tags the end tag is required. Thus, the following is legal.

```
<p>Here is a paragraph.</p><p>Here is another.</p>
```

However, this is not legal.

```
<p>Here is a paragraph.<p>Here is another.
```

5.1.2 *Mathematica* Support for XML

This section gives a very brief introduction to the *Mathematica* tools for working with XML; more information is available in the online documentation. One important point is that XML is suitable for holding structured data, which also applies to *Mathematica* expressions (the basic data type of *Mathematica*). This makes it easy to import XML data into *Mathematica*.

The following is a simple example.

```
In[1]:= xml =
          "<?xml version=\"1.0\"?>\n  <library>\n  <book>\n   <
            title>A New Kind of Science</title><author>Stephen Wolfram</
            author>\n  </book>\n  <book> \n   <title>The Lord of the Rings</
            title> <author>J.R.R. Tolkien</author>\n  </book>\n</library>";
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

This XML can be imported into *Mathematica,* which represents it with Symbolic XML. Because of the nature of *Mathematica* expressions, Symbolic XML is a *Mathematica* native form of XML that is isomorphic to textual XML.

```
In[2]:= sym = ImportString[ xml, "XML"]

Out[2]= XMLObject[Document][{XMLObject[Declaration][Version → 1.0]}, XMLElement[library,
          {}, {XMLElement[book, {}, {XMLElement[title, {}, {A New Kind of Science}],
             XMLElement[author, {}, {Stephen Wolfram}]}],
           XMLElement[book, {}, {XMLElement[title, {}, {The Lord of the Rings}],
             XMLElement[author, {}, {J.R.R. Tolkien}]}]}], {}]
```

You can use standard *Mathematica* programming features to process Symbolic XML; for example, this extracts all the authors.

```
In[3]:= Cases[sym, XMLElement[ "author", a_, {d_}] → d, Infinity]

Out[3]= {Stephen Wolfram, J.R.R. Tolkien}
```

```
In[4]:= newSym = sym /. XMLElement[ t_, a_, {d_}] → XMLElement[t, a, {ToLowerCase[d]}]

Out[4]= XMLObject[Document][{XMLObject[Declaration][Version → 1.0]}, XMLElement[library,
          {}, {XMLElement[book, {}, {XMLElement[title, {}, {a new kind of science}],
             XMLElement[author, {}, {stephen wolfram}]}],
           XMLElement[book, {}, {XMLElement[title, {}, {the lord of the rings}],
             XMLElement[author, {}, {j.r.r. tolkien}]}]}], {}]
```

Here new XML output is generated by modifying the original input.

```
In[5]:= ExportString[newSym, "XML"]

Out[5]= <?xml version='1.0'?>
        <library>
         <book>
          <title>a new kind of science</title>
          <author>stephen wolfram</author>
         </book>
         <book>
          <title>the lord of the rings</title>
          <author>j.r.r. tolkien</author>
         </book>
        </library>
```

This type of transformation can of course be done in other ways. For example, the use of XSLT stylesheet technology provides one way. However, there is an overhead to setting up an XSLT stylesheet to make the transformation. The use of *Mathematica*, with its uniform programming principles, is often a quick and simple way to get the task carried out.

There are many more features of the *Mathematica* XML tools, for example, working with attributes, entities, namespaces, validation, and CDATA. More information is available from the *Mathematica* documentation.

©1988-2006 Wolfram Research, Inc. All rights reserved.

5.1.3 web*Mathematica* XML Applications

Many web*Mathematica* applications involve generating HTML to be read by browsers. However, the output from a web*Mathematica* site may not go to a browser; it may involve some data to be read by an application that will then do further processing. This section will study an example that shows how this can be done.

The source for this example is in `webMathematica/Examples/XML/Phone.jsp`. It also uses an XML file `webMathematica/Examples/XML/phone.xml`. If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/XML/Phone.jsp. (You may have some other URL for accessing your server.)

We first see the XML data.

```
<?xml version="1.0"?>

<EmployeeList>
    <Person Name="Tom Jones" Email="tomj" Phone="235-1231" />
    <Person Name="Janet Rogers" Email="jrogers" Phone="235-1129" />
    <Person Name="Bob Norris" Email="bobn" Phone="235-1237" />
    <Person Name="Kit Smithers" Email="ksmit" Phone="235-0729" />
    <Person Name="Jamie Lemay" Email="jlemay" Phone="235-6393" />
</EmployeeList>
```

The contents of `Phone.jsp` are shown below.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<msp:evaluate>
    xml = Import[ ToFileName[ MSPPageDirectory[], "phone.xml"],
    "XML"] ;
    If[ MSPValueQ[ $$name],
        patt = "*" <> $$name <> "*";
        xml = DeleteCases[xml,
            XMLElement[ "Person", {___, "Name"->n_/;!StringMatchQ[n,
    patt],___},_ ], Infinity]] ;
    MSPReturn[ ExportString[ xml, "XML"], "text/xml"]
</msp:evaluate>
```

This example first imports the XML file into *Mathematica*. It uses the command `MSPPageDirectory` because the XML data is located in the same directory as `Phone.jsp`. It then checks to see if a parameter `name` was sent. If this is the case, then it uses this to discard XML elements that do not match this name. You should be able to see the operation of this parameter with a URL such as http://localhost:8080/webMathematica/Examples/XML/-Phone.jsp?name=T. (You may have some other URL for accessing your server.)

Of course, you may want to use this XML data for further processing. If you have a system that is XML-aware, this is quite straightforward. One useful application that is XML-aware is of course *Mathematica*. For example, the following will call your web*Mathematica* site and retrieve the information.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
In[1]:= XML`Parser`XMLGet[
         "http://localhost:8080/webMathematica/Examples/XML/Phone.jsp"]

Out[1]= XMLObject[Document][{XMLObject[Declaration][Version → 1.0], XMLObject[Comment][
         This is a demonstration XML file that is used as an example
         by webMathematica. The example demonstrates how to return
         XML from a webMathematica site.
        ]}, XMLElement[EmployeeList, {},
          {XMLElement[Person, {Name → Tom Jones, Email → tomj, Phone → 235-1231}, {}],
           XMLElement[Person,
            {Name → Janet Rogers, Email → jrogers, Phone → 235-1129}, {}],
           XMLElement[Person, {Name → Bob Norris, Email → bobn, Phone → 235-1237}, {}],
           XMLElement[Person, {Name → Kit Smithers, Email → ksmit, Phone → 235-0729}, {}],
           XMLElement[Person,
            {Name → Jamie Lemay, Email → jlemay, Phone → 235-6393}, {}]}], {}]
```

You may even wish to use this in a *Mathematica* program.

```
In[2]:= Contact[query_String] :=
         Cases[XML`Parser`XMLGet[
           "http://localhost:8080/webMathematica/Examples/XML/Phone.jsp?name=" <>
            query], XMLElement["Person", x_List, {}] :→ x, Infinity]

In[3]:= Contact[ "Tom"]

Out[3]= {{Name → Tom Jones, Email → tomj, Phone → 235-1231}}
```

Of course your client could be written in some system other than *Mathematica*, such as Visual Basic, Python, or Java.

5.2 MathML

MathML is designed to allow mathematical, scientific, and other technical information to be served, received, and processed on the World Wide Web. It is an official recommendation of the World Wide Web Consortium (W3C) working group on mathematics. Users of web*Mathematica* can benefit from MathML in a number of ways. They can use MathML for documents that contain a mixture of mathematics and text, they can generate MathML dynamically on their web*Mathematica* site, and they can use a MathML entry mechanism to enter mathematical notation into their web browser and send this to web*Mathematica* for computation.

Wolfram Research has long been involved in the development of MathML, both as a founding member of the mathematics working group of the W3C and as the host of the first two official MathML conferences in 2000 (http://www.mathmlconference.org/2000) and 2002 (http://www.mathmlconference.org/2002). *Mathematica* contains many features for working with MathML and there is a strong relationship between the *Mathematica* typesetting system and MathML.

One resource for learning more about MathML is the Wolfram Research sponsored web site, http://www.mathmlcentral.com. A section describing the evolution of MathML and some of the issues involved in developing a mathematical language suitable for a computation system such as *Mathematica* is found at http://www.mathmlcentral.com/history.html.

If you are not interested in the specific details of how MathML works and just want to use MathML in your output, then you should go to the sections Generating MathML and Sending MathML.

©1988-2006 Wolfram Research, Inc. All rights reserved.

5.2.1 Embedding MathML in Web Documents

This section discusses how documents can be written that mix both mathematics and text. These documents are written in XML format and use both MathML and XHTML (the XML compliant form of HTML). web*Mathematica* contains functions that do all of this automatically, so you do not need to read this unless you wish to learn more about the details of how browsers support MathML.

XHTML

XHTML is an XML compliant form of HTML, available as an official W3C recommendation, http://www.w3.org/-MarkUp. It is very similar to HTML, except that for a document to be valid it must follow the rules of XML. (Some of these were described in the previous section.) To use documents that mix mathematics and text, XHTML is required. Use of XHTML is needed anyway, since the W3C intends that HTML will not be developed further.

The sample XHTML document illustrated below is very similar to HTML, except for the initial XML declaration and the DTD reference. The latter can be used by an XML parser to validate that the input document is indeed valid XHTML. This demonstrates one of the benefits of XML technology. That is, a parser can validate a document, checking details such as the different tags being in the correct places and holding the correct number of arguments, without specializing in the particular flavor of XML. The reference to the DTD is not required; however, it is necessary if the document is to be validated.

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC
        "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Basic XHTML Document</title>
</head>
<body>
<h1>XHTML</h1>
<p>This is a basic XHTML document.</p>
</body>
</html>
```

This document could be read by modern web browsers and would display in the expected fashion.

XHTML and MathML

To add mathematics and other technical notation to a text document, it is possible to write one document that contains both XHTML and MathML. A sample document follows.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0//EN"
                "http://www.w3.org/TR/MathML2/dtd/xhtml-math11-f.dtd" [
        <!ENTITY mathml "http://www.w3.org/1998/Math/MathML">
    ]>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Basic XHTML+MathML Document</title>
</head>
<body>
<h1>XHTML+MathML</h1>
<p>Here is a math expression.</p>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<msup>
<mi>x</mi>
<mn>2</mn>
</msup></math>
</body>
</html>
```

This could be read into a browser that provides native support for MathML and would be read as expected. Note the reference to a DTD that allows the embedding of MathML into XHTML to form an XHTML+MathML document.

Unfortunately, not all browsers support MathML natively. While Mozilla, Amaya, and the most recent versions of Netscape do give native support for MathML, Internet Explorer does not.

For MathML to work with Internet Explorer or older versions of Netscape, a plug-in mechanism must be used. The way to do this is explored in the next section.

Rendering XHTML and MathML Documents

The previous section showed how to embed MathML into XHTML, creating documents that mix text and mathematics. It also explained that this does not work with browsers that rely on a plug-in mechanism. This section shows how to write documents that will work in a wide range of browsers.

To support MathML in browsers using a plug-in mechanism, the document must use special tags that are relevant to the particular plug-in used. If the browser supports MathML natively, then no special tags are needed. Of course, an author does not want to produce different versions of each document specific to each rendering technology. The solution is to make use of XSLT stylesheet technology to convert the document in the browser before it is viewed. This automatically inserts any special tags that are needed for plug-ins. An XSLT stylesheet that implements this solution is available from the W3C Math site, http://www.w3.org/Math/XSL.

Here is a document that uses the MathML stylesheet.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
    href="http://www.w3.org/Math/XSL/mathml.xsl"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Basic XHTML+MathML Document</title>
</head>
<body>
<h1>XHTML+MathML</h1>
<p>Here is a math expression.</p>
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<msup>
<mi>x</mi>
<mn>2</mn>
</msup>
</math>
</body>
</html>
```

This document can be rendered by any browser that is supported by the stylesheet. At the current time this includes the following:

Windows:

- Internet Explorer 6.0 with the MathPlayer plug-in
- Internet Explorer 5.5 with the MathPlayer plug-in
- Netscape 7.0 (and later)
- Amaya (Presentation MathML only)
- Mozilla 1.0 (and later)

Macintosh:

- Netscape 7.0 (and later)
- Mozilla 1.1 (and later)

Linux/Unix:

- Netscape 7.0 (and later)
- Mozilla 1.0 (and later)
- Amaya (Presentation MathML only)

See http://www.w3.org/Math/XSL for updates. Information on specific rendering technologies is also given in a later section.

©1988-2006 Wolfram Research, Inc. All rights reserved.

By using an absolute reference to the stylesheet, documents that use the stylesheet found on the W3C site can be moved from one server to another or saved locally and continue to work. One issue with an absolute stylesheet reference is that Internet Explorer may, according to its configuration, give a warning or even reject the stylesheet altogether (leading to a failure to render the MathML). This can be solved with a relative reference to the stylesheet and by placing a copy of the stylesheet on the same server as the document. For example, the document can start as follows.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
    href="/webMathematica/Resources/XSL/mathml.xsl"?>
```

This means that the stylesheet will be found at the URL `/webMathematica/Resources/XSL/mathml.xsl` relative to the root of the server from which the document is being retrieved. If a server chooses to do this, it will work well with Internet Explorer, but it will be necessary to ensure that the server has an up-to-date version of the stylesheet. It will also mean that documents will not be quite so portable when moved from one server to another.

Note that the XHTML+MathML document shown above that uses the MathML stylesheet does not contain a DOCTYPE declaration. This is, of course, a limitation because the document cannot now be validated. Another consequence is the XML system that renders it will not be aware of any special entity names. The DTD is missing because Internet Explorer does not accept all the entities in the MathML DTD. The solution is to use MathML which refers to numerical rather than named entities.

Here is an example that uses a named entity reference, `⁡`.

```
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<mrow>
<mi>sin</mi>
<mo>&af;</mo>
<mrow>
<mo>(</mo>
<mi>x</mi>
<mo>)</mo>
</mrow>
</mrow>
</math>
```

This example uses the numerical value `⁡`. It is the preferred form.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<mrow>
<mi>sin</mi>
<mo>&#8289;</mo>
<mrow>
<mo>(</mo>
<mi>x</mi>
<mo>)</mo>
</mrow>
</mrow>
</math>
```

If you want to find the numerical value for any character, you can use the *Mathematica* function `ToCharacterCode` to generate the numerical value, and `BaseForm` to generate the hexadecimal form. For example, the unicode value of a capital Alpha can be found as follows.

```
In[1]:= BaseForm[ToCharacterCode["A"], 16]

  Out[1]//BaseForm=
       {391_16}
```

Further information on the appropriate numerical values can be found at the MathML site, http://www.w3.org/TR/-MathML2/chapter6.html as well as at the unicode site, http://www.unicode.org.

5.2.2 Generating MathML from web*Mathematica*

Certain web*Mathematica* applications generate results that contain mathematical expressions suitable for formatting with MathML. This section shows how to generate MathML with web*Mathematica* and take advantage of the rendering techniques described in the previous section.

The main web*Mathematica* documentation describes how MathML can be generated with `MSPFormat` using a format style of `MathMLForm`. The following will format the expression `expr` into MathML.

```
<msp:evaluate>
    MSPFormat[ expr, MathMLForm]
</msp:evaluate>
```

MathML comes in two different varieties: presentation MathML specifies the appearance of the MathML whereas content MathML attempts to specify what the MathML means. Since MathML contains no general extension mechanism, the amount of information that can be encoded with content MathML is limited. However, if presentation MathML is generated from *Mathematica*, it will always work when sent back to *Mathematica*.

It is also possible to use other formatting styles, such as `StandardForm` or `TraditionalForm`, in which case the format type `RawMathML` should be selected, as shown here.

The following shows how to generate presentation MathML.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<msp:evaluate>
    MSPFormat[ expr, TraditionalForm, PresentationMathML]
</msp:evaluate>
```

The following generates content MathML.

```
<msp:evaluate>
    MSPFormat[ expr, TraditionalForm, ContentMathML]
</msp:evaluate>
```

Tools for working with MathML typically support both content and presentation.

MathML Integrate Example

This example JSP uses the MathML stylesheet. The page is actually a combination of two JSPs, `IntegrateForm.jsp` and `IntegrateXSLT.jsp`, that use JavaScript. They are closely modeled on the standard web*Mathematica* examples PlotScript.jsp and PlotScript1.jsp. The source for these MathML examples is available in `webMathematica/Examples/MathML`. If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/MathML/IntegrateForm.jsp . (You may have some other URL for accessing your server.)

First, here is the source for the input page, `IntegrateForm.jsp`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<html>
<head>
<script>
    function integrate(f)
    {
        win = window.open( "IntegrateXSLT?fun=" + f.fun.value,
    "integrate",

    "toolbar=none,resizeable=yes,width=450,height=350");
    }
</script>
<title>Integrate Tool</title>
</head>
<body bgcolor="#ffffff" >
<h1>Integrate Tool</h1>
<form action="IntegrateXSLT" method="post">

<p>
Enter a function:
<msp:allocateKernel>
<input type="text" name="fun"  size="24" value =
   "<msp:evaluate> MSPValue[ $$fun, "Sin[x]^2"] </msp:evaluate>"
>
</msp:allocateKernel>
</p>
<br/>
<input type="button" value="Integrate" onClick="integrate(this.form)"
    />
</form>
</body>
</html>
```

This is standard HTML. When the input button is clicked, a JavaScript function is called that extracts the input from the input field and calls the JSP `IntegrateXSLT.jsp`, which then opens in a new window. The contents of `IntegrateXSLT.jsp` are shown below.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
    href="/webMathematica/Resources/XSL/mathml.xsl"?>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Integrate Result</title>
</head>
<body>
<msp:allocateKernel>
<msp:evaluate>
    MSPPageOptions[ "ContentType" -> "text/xml"];
    res = Null;
    If[ MSPValueQ[ $$fun],
        fun = MSPToExpression[ $$fun];
        int = Integrate[ fun, x];
        If[ Head[int ] === Integrate, int = "Unknown"]] ;
</msp:evaluate>
<p>
Integration of a function, formatting into MathML.
</p>
<table border="2" rules="all">
<thead>
    <tr>
    <th>Function</th><th>Integral</th>
    </tr>
</thead>
    <tr>
    <td align="center"><msp:evaluate> MSPFormat[ fun,
    MathMLForm]<msp:evaluate></td>
    <td align="center"><msp:evaluate> MSPFormat[ int,
    MathMLForm]<msp:evaluate></td>
    </tr>
</table>
</msp:allocateKernel>
</body>
</html>
```

This uses the MathML stylesheet, which here is assumed to be installed in the web*Mathematica* web application in the directory `XSL`. The output content type is set to `text/xml`, and the necessary computation in *Mathematica* is carried out.

When this example works, it might be interesting to use the View Source menu of your browser. It should be noted how the MathML flows naturally with the XHTML. Also note how the document does not state the physical size of each mathematical expression. This is very useful because the size will only be known accurately when the document is rendered in the browser.

©1988-2006 Wolfram Research, Inc. All rights reserved.

The actual example code delivered with web*Mathematica* is a little more complicated since it contains alternatives for rendering directly with MathPlayer and for generating XHTML+MathML. However, the one shown above that uses the MathML stylesheet is the most general solution. The others are included in the example for demonstration purposes.

5.2.3 Sending MathML to web*Mathematica*

Many of the web*Mathematica* examples provide XHTML forms for users to enter input. This imposes certain limitations; for example, the input must use a one-dimensional syntax and cannot really use special characters. Further, it is often desirable to use special palettes to enter templates to be filled in. Although these are features provided by the *Mathematica* front end, they are not available in a web browser.

Although it is not possible to provide an alternative input mechanism in a pure browser such as Internet Explorer, more powerful input features are available with a plug-in. One suitable plug-in is the Input Control, included in the WebEQ suite of tools. WebEQ is a suite of tools for building web pages that involve dynamic math; it is described further in the Appendix: MathML Technology WebEQ.

MathML Input Example

This shows an example JSP that uses WebEQ for input. It is actually a combination of two JSPs, `IntegrateWebEQ.jsp` and `IntegrateXSLT.jsp`, that use JavaScript. The second JSP was also used in the previous example and renders MathML output. The two JSPs are closely modeled on the standard web*Mathematica* examples PlotScript.jsp and PlotScript1.jsp. If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/MathML/IntegrateWebEQ.jsp . (You may have some other URL for accessing your server.) Note that this assumes your browser has an installation of WebEQ. If you do not have WebEQ it will not work.

The text of `IntegrateWebEQ.jsp` is shown below.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<html>
<head>
<script>
    function integrate(f)
    {
        win = window.open( "IntegrateOutput?fun=" + f.fun.value,
    "integrate",

    "toolbar=none,resizeable=yes,width=450,height=350");
    }
</script>
<title>Integrate Tool</title>
</head>
<body bgcolor="#ffffff" >
<h1>Integrate Tool</h1>
<p>
Enter a function:
<msp:allocateKernel>
<applet code="webeq3.editor.InputControl" height=150 width=400
name="editor">
<param name="toolbar"
    value=",<tb><btn>SUP</btn><btn>FRAC</btn><btn>SQRT</btn>
    <btn>SIN</btn><btn>COS</btn><btn>TAN</btn></tb>">
<param name=eq value=
"<msp:evaluate> If[MSPValueQ[$$mml],
    $$mml,"<math><mrow></mrow></math>"] </msp:evaluate>">
</applet>
</msp:allocateKernel>
</p>
<br/>
<form action="IntegrateXSLT" method="post">

<input type="button" value="Integrate" onClick="integrate()" />
</form>
</body>
</html>
```

This is similar to IntegrateForm.jsp, except that WebEQ is used for entering input instead of an XHTML `input` element.

One of the key issues is that `IntegrateXSLT.jsp` works both with MathML input from WebEQ and text input from a form because the input processing function `MSPToExpression` accepts both MathML and text input. `MSPToExpression` also provides a security check on its argument, so it prevents dangerous unfiltered commands from being processed by *Mathematica*.

Any client machine that uses this example will need an installation of WebEQ. Information on how to obtain and install WebEQ is found at http://www.dessci.com/webmath/webeq/features.stm. In addition, the example could easily be modified so that a WebEQ archive was delivered from the server.

©1988-2006 Wolfram Research, Inc. All rights reserved.

5.3 SVG

SVG is a language for describing two-dimensional graphics in XML. Like MathML it is an official recommendation of the W3C, http://www.w3.org/Graphics/SVG/. It provides a number of benefits for users of web*Mathematica*. First, since it is a vector-based format the results often have a higher quality than is typically the case with image formats. This is very much the case when considering print output. Secondly, for many types of image, the actual file size is often quite small especially compared with image formats. Thirdly, it supports a number of dynamic and interactive features. *Mathematica* can generate SVG from graphics and this section will give some examples of web usage involving SVG. One thing to be noted is that any examples will require that your browser supports SVG. Ways to do this include the use of the Amaya browser, http://www.w3.org/Amaya/, which provides native support, or the Adobe plug-in, http://www.adobe.com/svg.

A utility package is provided with web*Mathematica* that supports adding the necessary tags to hook into the Adobe plug-in. This section will give some simple examples of the use of this package with web*Mathematica*.

5.3.1 Plotting with SVG

The source for this example is in `webMathematica/Examples/SVG/Plot.jsp`. It is closely related to the basic example, `Plot.jsp`. If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/SVG/Plot.jsp. (You may have some other URL for accessing your server.)

An extract of the source follows.

```
<form action="Plot.jsp" method="post">
<msp:allocateKernel>
<msp:evaluate>
    Get["MSP`SVG`"];
</msp:evaluate>
Enter a function:
<input type="text" name="fun"  size="24" value =
   "<msp:evaluate> MSPValue[ $$fun,  "Sin[x]^2"] </msp:evaluate> "
>
Enter a number:
<input type="text" name="x1"  size="24" value =
   "<msp:evaluate>  MSPValue[ $$x1,  "10"] </msp:evaluate> ">
<br>
<br>
<msp:evaluate>
    MSPBlock[ {$$fun, $$x1},
        SVGShow[ Plot[$$fun, {x,0,$$x1}]]]
</msp:evaluate>
<hr>
<input type="submit" name="btnSubmit" value="Evaluate">
</msp:allocateKernel>
</form>
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

This is very similar to the basic example Plot.jsp. The differences between the two are the loading of the SVG support package and the use of the SVG plotting function SVGShow. If this works correctly, you may wish to use some of the features that the Adobe plug-in provides.

5.3.2 SVG Animations

SVG supports a number of animation and interaction features. This example will demonstrate the use of SVG animations.

The source for this example is in webMathematica/Examples/SVG/NDSolvePlot.jsp. If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/web-Mathematica/Examples/SVG/NDSolvePlot.jsp. (You may have some other URL for accessing your server.)

The source contains an HTML form that sets up a number of input fields to collect the equation, starting and ending points, as well as initial conditions. These are then fed to a function that solves the differential equation and returns a plot of the result formatted as SVG. The SVG is then displayed with the function SVGDisplay, which is defined in the package MSP`SVG`. The code, which inserts the plot, is shown below.

```
<msp:evaluate>
    MSPBlock[ {$$eqn, $$t0, $$init1, $$init2, $$t1},
        svg = NDSolveToSVG[ $$eqn, {$$init1, $$init2}, {$$t0,$$t1}] ;
        SVGDisplay[ svg, {400, 300}]]
</msp:evaluate>
```

The actual definition of the function that creates the SVG is shown below. This solves the differential equation and generates a plot of the result. It then generates Symbolic XML, which represents the SVG of the result, using the function XML`SVG`GraphicsToSymbolicSVG. Next, it finds the points that represent the plot of the result and uses these to form an SVG animation of a red ball, which moves along these points. This animation is inserted into the SVG to form a new result, which is returned to be plotted.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
NDSolveToSVG[ eqn_, init_, lims_List]:=
    Module[ {sol, dep, t, int, t0, t1},
        {o,dep,t} = EquationToVariables[ eqn] ;
        {t0, t1} = lims ;
        sol=NDSolve[Append[init,eqn],dep,{t,t0,t1}];
        {int0, int1} = Part[dep /. First[ sol],1,1];
        If[ t0 < int0, t0 = int0];
        If[ t1 > int1, t1 = int1];
        p=ParametricPlot[ {dep[t],dep'[t]} /. sol,{t,t0,t1},
    ImageSize -> 400];
        xml = XML`SVG`GraphicsToSymbolicSVG[p];
        pts="M"<>
                First[Cases[ xml,
                        XMLElement["polyline",{"fill" -> _,"points"
    -> x_},_]->x, Infinity]];
        newElem=XMLElement["circle",
                {"cx"->"0","cy"->"0","r"->".1","fill"->"red",
    "stroke"->"blue",
                    "stroke-width"->"0.01"},
                {XMLElement[

    "animateMotion",{"dur"->"6s","repeatCount"->"indefinite",
                    "rotate"->"auto", "path" -> pts},{}]}];
        newXML=xml/.x:XMLElement["polyline",___] ->
    Sequence[x,newElem] ;
        ExportString[newXML,"XML"]
    ]
```

There are a number of other ways of obtaining interactive results with SVG. For example, JavaScript can interact with and manipulate the SVG tree thereby supporting interactive features such as pop-ups when the mouse is moved over a graphic.

5.4 HTML Formatting

One of the advantages of the HTML templating technique that web*Mathematica* provides is that there is often little need to try and generate HTML formats with *Mathematica* programs. In fact many of the HTML formatting issues can be left to web designers who can use their standard tools. However, sometimes it is useful to apply some HTML formatting functions to *Mathematica* expressions. This is particularly the case for HTML tables. In order to allow this, an HTML utility package is provided with web*Mathematica* that supports table formatting functions. This section will explore the use of this HTML formatting. A more general discussion of output is available in the section on Evaluation Formatting.

Remember that if you want to return HTML that is not generated by the HTML package, you should construct your own string of HTML and return this as shown in the example below.

```
<msp:evaluate>
    StringJoin[ "<b>", ToString[ x], "</b>"]
</msp:evaluate>
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

5.4.1 The HTML Functions

The HTML functions are contained in a package, MSP`HTML`, which is part of the web*Mathematica* layout. Since the package is loaded when web*Mathematica* starts there is no need to load the package manually. However, if you wish to use it in *Mathematica* outside of web*Mathematica*, you will need to copy the package into your AddOns/Applications directory, described in a previous section.

HTMLTableForm

As explained above, the MSP`HTML` package is available for web*Mathematica* and can be installed into regular *Mathematica*. It can then be loaded as shown below.

```
In[1]:= Needs["MSP`HTML`"]
```

The function HTMLTableForm takes an input and formats it into an HTML table.

```
In[2]:= HTMLTableForm[{{a, b, c}, {d, e, f}}]

Out[2]= <table border='1'>
         <tr>
          <td>a</td>
          <td>b</td>
          <td>c</td>
         </tr>
         <tr>
          <td>d</td>
          <td>e</td>
          <td>f</td>
         </tr>
        </table>
```

It takes a TableHeadings option, which works similar to that of TableForm.

```
In[3]:= HTMLTableForm[{{a, b, c}, {d, e, f}}, TableHeadings → {{r1, r2}, {c1, c2, c3}}]
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
Out[3]= <table border='1'>
         <th/>
         <th>c1</th>
         <th>c2</th>
         <th>c3</th>
         <tr>
          <th>r1</th>
          <td>a</td>
          <td>b</td>
          <td>c</td>
         </tr>
         <tr>
          <th>r2</th>
          <td>d</td>
          <td>e</td>
          <td>f</td>
         </tr>
        </table>
```

If you wish to apply special formatting to each element, you can provide a formatting function as a second element. The formatting function must return a string. Here every element is formatted into MathML.

```
In[4]:= HTMLTableForm[ {{x^2, Sin[x]}}, ExportString[#, "MathML"] &]
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
Out[4]= <table border='1'>
         <tr>
          <td><math xmlns='http://www.w3.org/1998/Math/MathML'>
         <semantics>
          <msup>
           <mi>x</mi>
           <mn>2</mn>
          </msup>
          <annotation-xml encoding='MathML-Content'>
           <apply>
            <power/>
            <ci>x</ci>
            <cn type='integer'>2</cn>
           </apply>
          </annotation-xml>
         </semantics>
        </math></td>
          <td><math xmlns='http://www.w3.org/1998/Math/MathML'>
         <semantics>
          <mrow>
           <mi>sin</mi>
           <mo>&#8289;</mo>
           <mo>(</mo>
           <mi>x</mi>
           <mo>)</mo>
          </mrow>
          <annotation-xml encoding='MathML-Content'>
           <apply>
            <sin/>
            <ci>x</ci>
           </apply>
          </annotation-xml>
         </semantics>
        </math></td>
         </tr>
        </table>
```

The default formatting function for `HTMLTableForm` is `HTMLFormat`, which is described below.

Any string arguments to `HTMLTableForm` are assumed to be already formatted and no more formatting is applied. This allows it to take the output of other MSP functions such as `MSPShow` or `MSPFormat`.

HTMLFormat

As explained above, the `MSP`HTML`` package is available for web*Mathematica* and can be installed into regular *Mathematica*. It can then be loaded as shown below.

```
In[1]:= Needs["MSP`HTML`"]
```

The function `HTMLFormat` provides some useful formatting functions into HTML. It is suitable for formatting small expressions such as numbers as shown below.

```
In[2]:= HTMLFormat[x^2]
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
Out[2]= x<sup>2</sup>

In[3]:= HTMLFormat[10.!]

Out[3]= 3.6288 10<sup>6</sup>
```

It is less suitable for formatting large expressions, since everything will come out in `InputForm`.

```
In[4]:= Nest[ 1 / (1 - #) &, x, 5]

Out[4]=
                1
  -------------------------
                1
  1 - -------------------
                1
      1 - ---------
                1
          1 - ---
              1-x

In[5]:= HTMLFormat[% ]

Out[5]= 1 - 1 - 1 - 1 - 1 - 1 - 
          x<sup>-1</sup><sup>-1</sup><sup>-1</sup><sup>-1</sup><sup>-1</sup>
```

For larger expressions it is recommended to use one of the versions of the formatting function `MSPFormat` to gain a result in an image format or MathML.

HTMLSelect

As explained above, the `MSP`HTML`` package is available for web*Mathematica* and can be installed into regular *Mathematica*. It can then be loaded as shown below.

```
In[1]:= Needs[ "MSP`HTML`"]
```

The function `HTMLSelect` provides a useful way to generate select tags with web*Mathematica*. It takes a list of the different options and the name to be used when the selection is submitted. Its operation is shown below.

```
In[2]:= HTMLSelect[ {"a", "b", "c"}, "arg1"]

Out[2]= <select name='arg1'>
         <option value='1'>a</option>
         <option value='2'>b</option>
         <option value='3'>c</option>
        </select>
```

It is also possible to set selections by using the option `SelectedOptions`. In this example, the option labeled 'a' will be selected.

```
In[3]:= HTMLSelect[ {"a", "b", "c"}, "arg1", SelectedOptions → a]

Out[3]= <select name='arg1'>
         <option value='1' selected='selected'>a</option>
         <option value='2'>b</option>
         <option value='3'>c</option>
        </select>
```

By default the values for the option tags are chosen automatically. It is also possible to set these with an argument.

```
In[4]:= HTMLSelect[ {"a", "b", "c"}, {"x", "y", "z"}, "arg1"]
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
Out[4]= <select name='arg1'>
         <option value='x'>a</option>
         <option value='y'>b</option>
         <option value='z'>c</option>
        </select>
```

The option SelectedValues can be used to set a selection based on the values.

```
In[5]:= HTMLSelect[ {"a", "b", "c"}, {"x", "y", "z"}, "arg1", SelectedValues → y]

Out[5]= <select name='arg1'>
         <option value='x'>a</option>
         <option value='y' selected='selected'>b</option>
         <option value='z'>c</option>
        </select>
```

The selection options can take a list of values to set a multiple selection.

```
In[6]:= HTMLSelect[ {"a", "b", "c"}, {"x", "y", "z"}, "arg1", SelectedValues → {x, y}]

Out[6]= <select name='arg1'>
         <option value='x' selected='selected'>a</option>
         <option value='y' selected='selected'>b</option>
         <option value='z'>c</option>
        </select>
```

If no values are given, the SelectedValues option can use the numerical values.

```
In[7]:= HTMLSelect[ {"a", "b", "c"}, "arg1", SelectedValues → {3}]

Out[7]= <select name='arg1'>
         <option value='1'>a</option>
         <option value='2'>b</option>
         <option value='3' selected='selected'>c</option>
        </select>
```

HTMLCheckBox

As explained above, the MSP`HTML` package is available for web*Mathematica* and can be installed into regular *Mathematica*. It can then be loaded as shown below.

```
In[1]:= Needs[ "MSP`HTML`"]
```

The function HTMLCheckBox provides a useful way to generate an input checkbox tag with web*Mathematica*. It takes the name to use when the check box is submitted as an argument . Its operation is shown below.

```
In[2]:= HTMLCheckBox[ boxname]

Out[2]= <input type='checkbox' name='boxname'/>
```

If a second argument is given, this is used to determine whether or not the box is checked. In the following example the check box is checked.

```
In[3]:= HTMLCheckBox[boxname, 10 > 5]
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
Out[3]= <input type='checkbox' name='boxname' checked='checked'/>
```

5.4.2 web*Mathematica* Examples

A number of web*Mathematica* examples are provided that make use of the HTML formatting package. These are shown in this section.

Table Formatting

A first simple example is `Table.jsp`, the source for which is available in `webMathematica/Examples/HTML`. If you installed the web*Mathematica* webapp as described above, you should be able to connect to it via http://localhost:8080/webMathematica/Examples/HTML/Table.jsp. (You may have some other URL for accessing your server.)

A second example is `RegressTable.jsp`, the source for which is available in `webMathematica/Examples/HTML`. If you installed web*Mathematica* as described above, you should be able to connect to it via http://localhost:8080/webMathematica/Examples/HTML/RegressTable.jsp. (You may have some other URL for accessing your server.) A section of the contents are shown below.

```
<msp:evaluate>
   Needs["Statistics`LinearRegression`"];
   Needs["Statistics`Common`RegressionCommon`"];
</msp:evaluate>

<msp:evaluate>
   data = {{0.055, 90}, {0.091, 97}, {0.138, 107},
        {0.167, 124}, {0.182, 142}, {0.211, 150},
        {0.232, 172}, {0.248, 189}, {0.284, 209},
        {0.351, 253}};
   regress = Chop[ Regress[data, {1, x^2}, x], 10^-6];
   pTable = ParameterTable /. regress ;
   aTable = ANOVATable /. regress ;
   rSq = RSquared /. regress ;
   aSq = AdjustedRSquared /. regress ;
   eVar = EstimatedVariance /. regress ;
</msp:evaluate>

<br/>

<msp:evaluate>
    HTMLTableForm[ MSPShow[ ListPlot[ data, Frame -> True]],
     TableHeadings -> {"Data to be fitted"}]
</msp:evaluate>

<br/>

<msp:evaluate>
    HTMLTableForm[ HTMLTableForm[ pTable], TableAttributes -> {},
     TableHeadings->{"ParameterTable"}]
</msp:evaluate>
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

This shows how the packages are loaded. Note how the subpackage must be loaded as well. The data is assigned (typically this would be loaded in some dynamic fashion), and the regression analysis is carried out. Two uses of HTMLTableForm then follow. In the first, the result of MSPShow is put into a table with a heading. This is a convenient way to attach a border and heading to something. In the second, the parameter table, pTable, is put into a table. This table is itself put into another table to get a heading.

Select Formatting

An example of the use of HTMLSelect is in Select.jsp, the source for which is available in webMathematica/Examples/HTML. If you installed the web*Mathematica* webapp as described above, you should be able to connect to it via http://localhost:8080/webMathematica/Examples/HTML/Select.jsp. (You may have some other URL for accessing your server.) The source is shown below.

```
<form action="Select.jsp" method="post">

<msp:allocateKernel>

<msp:evaluate>
    days =
    {"Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Satu
    rday"};
    day = Null;
    If[ MSPValueQ[ $$daySelected],
        dayPT = MSPToExpression[ $$daySelected];
        day = Part[ days, dayPT]];
</msp:evaluate>

<msp:evaluate>
    HTMLSelect[ days, daySelected, day]
</msp:evaluate>

<msp:evaluate>
    If[ day =!= Null,
        dayPT = Mod[ dayPT+1,7,1] ;
        "The day after the day selected is " <> Part[ days, dayPT]]
</msp:evaluate>

</msp:allocateKernel>
<br>
<br>

<input type="submit" name="btnSubmit" value="Evaluate">

</form>
```

In this example, the input parameter $$daySelected is inspected and used to determine which day was selected. The second evaluation actually puts down the select tag, showing how easy this is. The last evaluation computes the day after the day selected by incrementing the dayPT variable and then takes its modulus with respect to 7 with an offset of 1.

©1988-2006 Wolfram Research, Inc. All rights reserved.

5.5 Using Java APIs

The purpose of web*Mathematica* is to allow *Mathematica* computations to be run by a web server. These computations will typically involve one of the many tasks for which *Mathematica* is well suited, such as numerical or symbolic computation. However, sometimes it is useful in a web*Mathematica* computation to call outside of *Mathematica* to gain some extra functionality. The most convenient way to do this is to make use of Java. Regular interactive *Mathematica* can call to Java very easily with the *J/Link* toolkit, and web*Mathematica* can do the same. More information on the concept of working with Java APIs and referencing Java objects from within *Mathematica* can be found in the *J/Link* documentation.

In web*Mathematica* two classes of calls can be distinguished: those to server related APIs and those to more general Java APIs. These are discussed in the following two sections.

5.5.1 Server APIs

Calls to server-specific APIs that govern the operation and details of a particular web*Mathematica* site are facilitated by definition of the following server objects.

`$ServletRequest`	The `HttpServletRequest` object for this request.
`$ServletResponse`	The `HttpServletResponse` object for this request.
`$ServletConfig`	The `ServletConfig` object for the server.

These are all Java object references maintained by *J/Link* and can be used in the typical *J/Link* fashion. `$ServletRequest` holds a reference to the servlet `HTTPServletRequest` object, `$ServletResponse` holds a reference to the servlet `HTTPServletResponse` object, and `$ServletConfig` holds a reference to the `ServletConfig` object. The various methods for these objects are documented as part of the servlet API and will be found in any reference to servlets. For example, the request object has a method `getRemoteAddr`, which can be used in an MSP as follows.

```
<msp:evaluate>
    $ServletRequest@getRemoteAddr[]
</msp:evaluate>
```

This will return the IP address of the client that sent the request and is equivalent to the CGI variable `REMOTE_ADDR`.

A more elaborate example is found in `Request.jsp`, the source for which is available in `webMathematica/Examples`. If you installed web*Mathematica* as described above, you should be able to connect to it via http://localhost:8080/webMathematica/Examples/Request.jsp. (You may have some other URL for accessing your server.) This example extracts names and values from the HTTP request.

Note that any Java object references created when processing a particular page will be released when the whole page finishes. Note that Java objects created during initialization of the kernel will not be removed, providing a mechanism to maintain Java objects that persist from one call to another. Despite the fact that Java objects are automatically released, it is strongly recommended that all Java objects are either created inside of a call to `Java`

©1988-2006 Wolfram Research, Inc. All rights reserved.

Block or use ReleaseObject explicitly. You can learn more about JavaBlock and ReleaseObject in the *J/Link* documentation.

5.5.2 Other Java APIs

There are many other Java APIs that can be used by web*Mathematica*. These include APIs for database connectivity, XML processing, speech generation, data format I/O, and calling via HTTP to other web services. All of these are readily available to web*Mathematica*. For information read the appropriate Java reference.

5.6 Data Loading and Computation

Mathematica contains a variety of data loading functions, available through the function Import. This supports many formats such as comma and tab-delimited text data, as well as more specialized formats for graphics, science, sound, and XML. In addition binary data can be loaded using the function Experimental`BinaryImport. If you find that your particular data is not well supported directly by *Mathematica*, it is possible that you may use a Java API to load the data, which was described in the previous section.

In order to develop data loading technology, it is probably a good idea to work in interactive *Mathematica* so that you understand how the data loading functions work. When you have done this, you can add data loading to your web applications. At this point you need to determine the source for your data. The following sections discuss some of the possibilities.

5.6.1 File I/O

If your data files are available to the file system of the computer on which your web*Mathematica* server runs, they can be read with a command like Import. For this to work the name and location of the data file must be specified. This can be done in several ways. One way involves placing the files into a directory and setting the full pathname of this directory. This suffers from the extreme disadvantage that if you change the name of the directory you will have to modify all your scripts that use this name. An improvement could be obtained by setting the name of the directory with an initialization parameter. This can be done in MSP.conf with the parameter KernelInitialization as shown in the following.

```
KernelInitialization=MyApplication`DataDirectory="C:\\Work\\Data"
```

This assigns the *Mathematica* symbol MyApplication`DataDirectory to "C:\Work\Data". Note the use of a full context name. This is necessary to prevent the symbol being cleared by the Kernel cleaning mechanism. This can then be loaded in a web*Mathematica* computation as shown in the following example.

```
<msp:evaluate>
    data = Import[ ToFileName[ MyApplication`DataDirectory,
     "file.dat"], "Table"];
</msp:evaluate>
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

An alternative is to place the data file into a directory that is on the *Mathematica* path setting `$Path`. This is the approach taken by the example `Data1.jsp`, which is shown below. Another alternative is to place the data file into the same directory as the script, and to use `MSPScriptDirectory[]`, as shown in the following.

```
<msp:evaluate>
    data = Import[ ToFileName[MSPScriptDirectory[], "file.dat"],
     "Table"];
</msp:evaluate>
```

This was used in the XML example `Phone.jsp`, which was discussed previously. It is particularly convenient to use `MSPScriptDirectory[]` since it means that data and scripts live in the same directory. Thus the entire web application can be moved from one server to another with a minimum of setting up. One disadvantage is that for JSPs the data file can be loaded by a direct request to the server; thus it should only be used if there is no specialized information present in the data file. This might be the case if only certain information was suitable to be used in a response to each request.

5.6.2 HTTP Upload

Another way to load data into a web*Mathematica* server is to send it from the client machine with the HTTP request. web*Mathematica* contains tools to support this with the function `MSPGetUploadFile`. This is demonstrated in the example, `Upload.jsp`, which is shown below.

5.6.3 Database Connectivity

DatabaseLink provides *Mathematica* with an industrial-strength, ready-made solution for integrating *Mathematica* with any standard SQL database. Integrated with *Mathematica* it provides a convenient bridge between SQL databases and web*Mathematica*.

DatabaseLink is based on Java Database Connectivity (JDBC) technology and so it fits very well with the Java technology on which web*Mathematica* is based. It has many useful and important features (listed at http://documents.wolfram.com/mathematica/Add-onsLinks/DatabaseLink/). One particularly useful feature for web*Mathematica* is that *DatabaseLink* contains the HSQL Database Engine (HSQLDB), a lightweight database. This means that if you do not already have a database or want to experiment with using one you do not have to set one up.

web*Mathematica* contains two examples of working with a database. These are `Database.jsp`, which is accessible via http://localhost:8080/webMathematica/Examples/Database/Database.jsp and `DatabaseTable.jsp`, which is accessible via http://localhost:8080/webMathematica/Examples/Database/DatabaseTable.jsp . Note that for these links to work you need to have installed web*Mathematica*, as described above, and also the *DatabaseLink* example databases, as described in its documentation. The example databases are installed into `$UserBaseDirectory`, and so you need to make sure this is `$UserBaseDirectory` for the user who is running web*Mathematica*.

©1988-2006 Wolfram Research, Inc. All rights reserved.

5.6.4 Web Services

Another way to find data for *Mathematica* is by contacting another web site. A particularly convenient way to do this is if the other web site provides the data as a web service. In this case *Mathematica* can use the *Web Services Package* (described at http://documents.wolfram.com/mathematica/Add-onsLinks/WebServicesPackage) to call the web site and use its services.

web*Mathematica* contains an example of calling to a web service. This is `WebServices.jsp`, which is accessible via http://localhost:8080/webMathematica/Examples/WebServices/WebServices.jsp . Note that for these links to work you need to have installed web*Mathematica*, as described above.

5.6.5 Data Examples

The following are a collection of examples of web*Mathematica* working with data.

Loading Data: Load.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/Data/Load.jsp . (You may have some other URL for accessing your server.)

This example shows how to load *Mathematica* packages and how to read a data file that is stored on the server. The data is processed with a data smoothing algorithm and then a couple of plots are made. The source for this page is in `webMathematica/Examples/Data/Load.jsp`. A section that shows the `form` tag is shown below.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<form action="Load.jsp" method="post">

<msp:allocateKernel>

<msp:evaluate>
    Needs[ "Graphics`MultipleListPlot`"];
    Needs[ "Statistics`DataSmoothing`"];
</msp:evaluate>

<msp:evaluate>
    data = Flatten[ N[ Import[ "Data/DataFile1.dat"]]];
    term = 4;
    If[ MSPValueQ[ $$term],
        term = MSPToExpression[ $$term]];
</msp:evaluate>



<msp:evaluate>
    dataSmooth = MovingAverage[ data, term];
    MSPShow[ MultipleListPlot[ data,dataSmooth,
            PlotJoined ->{False,True},
            SymbolShape -> Point,
            SymbolStyle -> {{PointSize[0.008],Hue[0]},
    {PointSize[0.001]}},
        ImageSize -> 600]]
</msp:evaluate>

<br />
<br />

Number of smoothing terms:
<input type="text" name="term" size="3"
    value="<msp:evaluate>MSPValue[ $$term, "3"]</msp:evaluate>"  />

</msp:allocateKernel>

<br />
<br />
<input type="submit" name="btnSubmit" value="Evaluate">
</form>
```

In this example, an `msp:evaluate` tag is used to load two *Mathematica* packages. It is good practice to always load packages in their own tag. The next `msp:evaluate` tag loads the data set `Data/DataFile1.dat` using the `Import` command. This will search for the data file on the *Mathematica* `$Path` and will find the `Data` directory inside of the `MSPScripts` directory, which by default is located in `webMathematica/WEB-INF`. It is worth looking inside of your web*Mathematica* web application and confirming that you can locate the data file. The example then computes a moving average and plots the original and the smoothed data. This is all placed inside of a `form` element so that it is possible to modify the number of terms.

One weakness of this example is that the data has to be loaded from the data file for every computation. It would be better to save the data somehow. In addition it might be useful to allow the data to be uploaded from the client. These will be explored in the sections that follow.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Uploading Data: Upload.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this example via http://localhost:8080/webMathematica/Examples/Data/Upload.html . (You may have some other URL for accessing your server.)

This example allows the user to enter a function to be integrated. The result is then formatted by the typesetting system and saved as an image. The source for this page is `webMathematica/Examples/Data/Upload.html` and `webMathematica/Examples/Data/Upload.jsp`. The contents of `Upload.html` are shown below.

```
<html>
<head>
<title>Upload File</title>
</head>
<body>
<form method=post enctype="multipart/form-data" action="Upload.jsp">
<p>
Enter a file to upload:
</p>
<input type="file" size=40 name="file">
<br />
<br />
<input type="reset" value="Clear">
<input type="submit" value="Submit">
</form>

<i>
To generate suitable data,
<a href="DataGenerate.jsp">click here</a>
</i>
</body>
</html>
```

It should first be noted that this is an HTML page; it contains no Java or *Mathematica* inserts. A servlet container will deliver HTML pages just as it delivers JSPs, and they can all be put together in the same directories inside of the webapp, which can also contain other files such as those containing images or movies. This file contains a single `form` element that is set up to submit a data file using a `enctype` attribute of `multipart/form-data` and an `input` element of type `file`. When the submit button is clicked, the form will be submitted along with the file to the `Upload.jsp`. A selection of `Upload.jsp` is shown below.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<msp:allocateKernel>

<msp:evaluate>
    Needs[ "Graphics`MultipleListPlot`"];
    Needs[ "Statistics`DataSmoothing`"];
</msp:evaluate>

<msp:evaluate>
    file = "FileName" /. MSPGetUploadFile[];
    data = Flatten[ N[ Import[ file, "Table"]]];
    term = 4;
</msp:evaluate>

<msp:evaluate>
    If[ StringQ[ file],
        dataSmooth = MovingAverage[ data, term];
        MSPShow[ MultipleListPlot[ data,dataSmooth,
            PlotJoined ->{False,True},
            SymbolShape -> Point,
            SymbolStyle -> {{PointSize[0.008],Hue[0]},
    {PointSize[0.001]}},
            ImageSize -> 600]]
</msp:evaluate>

</msp:allocateKernel>
```

In this selection we see how the MSP function `MSPGetUploadFile` is used to retrieve the name of the data file by which the data has been stored on the server. This filename can then be used to read the data as was done in the previous example and the computation can proceed. It should be noted that this solution is somewhat limited; the data is only available once for one computation, and no facility has been added to change a parameter, for example, to control the data smoothing. For this the data needs to be stored in a session variable. This is explored in the next example.

In this example a test was made to determine whether the filename was in fact a string. This checks that a data file has in fact been uploaded. A file would not be uploaded if the page `Upload.jsp` was visited directly instead of coming from a request to `Upload.html`.

`MSPGetUploadFile` returns a list of useful information, including the filename that is used on the server, the original filename used on the client, and the content-type. In a case where there are multiple files to be uploaded, `MSPGetUploadFile` will throw an exception. If you wish to upload more than one file, you can use `MSPGetUploadFileList`.

Session Storage of Data: Session.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/Data/Session.html . (You may have some other URL for accessing your server.)

©1988-2006 Wolfram Research, Inc. All rights reserved.

This example allows the user to enter a function to be integrated. The result is then formatted by the typesetting system and saved as an image. The source for this page is webMathematica/Examples/Data/Session.html, webMathematica/Examples/Data/Session.jsp, and webMathematica/Examples/Data/SessionProcess.jsp. The contents of Session.html are shown below.

```
<html>
<head>
<title>Upload File</title>
</head>
<body>
<form method=post enctype="multipart/form-data" action="Session.jsp">
<p>
Enter a file to upload:
</p>
<input type="file" size=40 name="file">
<br />
<br />
<input type="reset" value="Clear">
<input type="submit" value="Submit">
</form>

<i>
To generate suitable data,
<a href="DataGenerate.jsp">click here</a>
</i>
</body>
</html>
```

This is very similar to Upload.html, except that it calls the JSP Session.jsp when the form is submitted. Session.jsp is shown below.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<msp:allocateKernel>

<msp:evaluate>
    file = "FileName" /. MSPGetUploadFile[];
    data = Flatten[ N[ Import[ file, "Table"]]];
    MSPSessionVariable[ UploadedData, Null];
    UploadedData = data;
</msp:evaluate>

</msp:allocateKernel>

<jsp:forward page="SessionProcess.jsp" />
```

This short JSP reads the data that was uploaded, and then uses MSPSessionVariable to store the data in a session variable named UploadedData. This is a very convenient way to store data persistently from one HTTP request to another. The data is actually stored in the server, so that even if the *Mathematica* kernel was restarted the data would still be available. The storage uses what is known as an HTTP session and the server may use cookies or some other mechanism to actually store the data. This is how a shopping cart in an e-commerce web site works.

©1988-2006 Wolfram Research, Inc. All rights reserved.

As far as a developer of a web*Mathematica* web site is concerned, it is all very simple; just use the function `MSPSessionVariable`, giving it the name of the variable and an initial value. After the data has been read, there is a jsp:forward to the page that actually does the computations and plots, `Data3.jsp`. Often it is a very good design principle to divide the code over a number of separate pages that all do different tasks; if you develop pages that have large amounts of complicated code in them, then refactoring might improve your system. A selection of `SessionProcess.jsp` is shown below.

```
<form action="SessionProcess.jsp" method="post">

<msp:allocateKernel>

<msp:evaluate>
    Needs[ "Graphics`MultipleListPlot`"];
    Needs[ "Statistics`DataSmoothing`"];
</msp:evaluate>

<msp:evaluate>
    MSPSessionVariable[ UploadedData, Null];
    term = 4;
    If[ MSPValueQ[ $$term],
        term = MSPToExpression[ $$term]];
</msp:evaluate>

<msp:evaluate>
    If[ UploadedData =!= Null,
        dataSmooth = MovingAverage[ UploadedData, term];
        MSPShow[ MultipleListPlot[ UploadedData,dataSmooth,
            PlotJoined ->{False,True},
            SymbolShape -> Point,
            SymbolStyle -> {{PointSize[0.008],Hue[0]},
    {PointSize[0.001]}},
        ImageSize -> 600]]]
</msp:evaluate>
<br /><br />
Number of smoothing terms:
<input type="text" name="term" size="3"
    value="<msp:evaluate>MSPValue[ $$term, "3"]</msp:evaluate>"  />

</msp:allocateKernel>
<br /><br />
<input type="submit" name="btnSubmit" value="Evaluate">
</form>
```

`SessionProcess.jsp` is very similar to `Load.jsp`; the main difference is the line that obtains the data. Instead of reading data from a file, it uses the session variable `UploadedData`, which was assigned to the data in the previous JSP, `Session.jsp`. If, by some chance, `UploadedData` was not previously defined, as might happen if `SessionProcess.jsp` is visited directly, then `UploadedData` will have the value `Null`. Here this just prevents any plot from being produced, but in general it might be used to transfer to an error page.

A further change to this section might be to store the data permanently on the server. For a simple data file, this could easily be done by using *Mathematica* file output operations. A more sophisticated application could use Java database connectivity to store the data and certain related information.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Session variables are discussed further in Advanced Topics: Variables.

Database Connections: Database.jsp

If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/Database/Database.jsp. (You may have some other URL for accessing your server.)

This example shows how to use *DatabaseLink* to connect to an SQL database. The database in question is one that is contained in HSQLDB, and for it to work the sample databases need to be installed. This is described in the documentation (http://documents.wolfram.com/mathematica/Add-onsLinks/DatabaseLink/). The databases are installed into `$UserBaseDirectory`, and so you need to make sure this is `$UserBaseDirectory` for the user who is running web*Mathematica*.

This example shows how you can open a connection to a database and make a query to a particular table based on an input parameter. The data is displayed with `HTMLTableForm`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<msp:evaluate>
    Needs[ "DatabaseLink`"];
</msp:evaluate>

<form action="Database.jsp" method="post">

<msp:evaluate>
limit = 6000;
If[ MSPValueQ[ $$limit],
    limit = MSPToExpression[ $$limit]];
</msp:evaluate>

Lower limit:
<input type="text" name="limit" size="24" value =
    "<msp:evaluate>MSPValue[ $$limit,  "6000"]</msp:evaluate>"
/>

<msp:evaluate>
    conn = OpenSQLConnection[ "publisher"];
    headings = {"TITLE_ID" , "LORANGE", "HIRANGE", "ROYALTY"};
    data=SQLSelect[conn, {"ROYSCHED"},
            headings,
            SQLColumn["LORANGE"] > limit];
</msp:evaluate>

<input type="submit" name="submitButton" value="Search">

</form>

<msp:evaluate>
    HTMLTableForm[ data, TableHeadings -> headings]
</msp:evaluate>

<!--
 The shutdown command is specific to the HSQL database that
 is used for these examples.
 -->

<msp:evaluate>
    SQLExecute[ conn, "SHUTDOWN"];
    CloseSQLConnection[ conn];
</msp:evaluate>
```

The code loads *DatabaseLink* and carries out some processing of an input variable, which is used to select data from the database. It opens a connection to the database and selects data using the input variable and then formats and prints the result. Finally, it shuts down the database and closes the connection. The shut down instruction is specific to HSQLDB and is necessary to make sure that the database can be used if the server is restarted.

©1988-2006 Wolfram Research, Inc. All rights reserved.

5.7 *Mathematica* Packages and Applications

web*Mathematica* provides a way to embed *Mathematica* code inside of HTML. If the amount of code is significant, it might be more convenient to place the code into a *Mathematica* package and then refer to the package. In addition, a script might be needed to make use of existing *Mathematica* packages or applications—one large group of *Mathematica* packages being the standard add-on packages. This section discusses how to work with *Mathematica* code and packages.

5.7.1 Loading Packages

It is relatively simple to add code that loads a package. Here is a simple example.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<html>
<head>
<title>Live 3D Plotting</title>
</head>

<body text="#171717" bgcolor = "#ffffff">
<html>
  <head>
    <title>ConvexHull Computation</title>
  </head>
  <body bgcolor="#FFFFFF">
    <h1>ConvexHull Computation</h1>
    <msp:allocateKernel>
    <msp:evaluate>Needs["DiscreteMath`ComputationalGeometry`"];
     </msp:evaluate>

    <msp:evaluate>ConvexHull[
     {{1,5},{4,1},{10,2},{5,4}}]</msp:evaluate>
    </msp:allocateKernel>
  </body>
</html>
```

Notice how the `<msp:evaluate>` tag uses `Needs` to load the package. An alternative, but less desirable, way to load a package is with `Get`. This is much less efficient since the package will be loaded each time the page is loaded. When `Needs` is used, the package is only loaded the first time.

An important detail is that one tag loads the package and another uses a function from the package. A tag that loads a package should not use functions that come from the package. If this happens, a shadowing symbol is created, which will mask the function you wish to use. You can read more about shadowing of symbols in *The Mathematica Book*. When a shadowing symbol is created, a warning message is issued; the text of messages can be obtained with MSPGetMessages. It will also be displayed in the log files if verbose logging is enabled. The sections on Logging and The Kernel Monitor discuss verbose logging and log files.

A final issue concerns the use of subcontexts in packages. This can be an issue with *Mathematica* application packages, which often arrange their symbols in subcontexts. For example, the image processing package uses

©1988-2006 Wolfram Research, Inc. All rights reserved.

subcontexts. (Note that this example will not work on your system if you do not have the ImageProcessing application package installed.)

```
In[1]:= Needs["ImageProcessing`"]

In[2]:= Context[ ImageRead]

Out[2]= ImageProcessing`ImageData`
```

Here the ImageRead function is defined in the ImageProcessing`ImageData` context. The implication for web*Mathematica* coding is that a Needs statement is required for all the contexts that contain symbols you wish to use directly inside of <msp:evaluate> tags. Thus, if you wish to use the ImageRead function, you must insert a Needs statement for the ImageProcessing` application and also for the context that contains the ImageRead function. An example follows.

```
<msp:evaluate>
     Needs["ImageProcessing`"];
     Needs["ImageProcessing`ImageData`"];</msp:evaluate>

 <msp:evaluate>
     ImageRead[ "datafile"]</msp:evaluate>
```

If you feel that packages are not being used correctly, it would be good to check that the contexts for symbols you are using have Needs statements. One way to check this would be to write a small page that checked the contexts of functions you were using.

```
<msp:evaluate>
    Needs["ImageProcessing`"];
    Needs["ImageProcessing`ImageData`"];</msp:evaluate>

<msp:evaluate>
    Context[ ImageRead]</msp:evaluate>
```

If you have installed the image processing application and run this script on your server, you should see Image Processing`ImageData` returned. This confirms that you have correctly loaded the function from the image processing application.

Similar issues apply if you use the Master mechanism to load packages. You will either require a Needs statement for the individual contexts, or refer to the full context name of the symbols. Of course, the former makes the Master mechanism not useful. An example of using full contexts is shown below.

```
<msp:evaluate>
    Needs["Graphics`Master`"];</msp:evaluate>

<msp:evaluate>
    color = Graphics`Color`Red </msp:evaluate>
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

5.7.2 Writing Packages

If you write any significant amount of your own code, it is a good idea to write it as a *Mathematica* package and load it into web*Mathematica*. This is particularly important for web*Mathematica* since you want to reduce the amount of *Mathematica* code that you have in your web*Mathematica* pages. There are a number of references that help with the process of writing a package; these are given in Links: *Mathematica* Packages. Instructions on how to load the package are given in the previous section; information on the location to place your package is given in the following section.

If you do not use the *Mathematica* package format, but instead use global definitions for your code, then you will need to load it every time the script is accessed. This is because of the post-processing that takes place when a script is accessed. It is recommended that you place code into a *Mathematica* package.

5.7.3 Installing Packages

When a package of *Mathematica* code is available, it needs to be installed in some location so that it can be used by web*Mathematica*. There are a number of ways this can be done; they are covered in the following sections.

web*Mathematica* Applications

web*Mathematica* provides an Applications directory, located in `webMathematica/WEB-INF/Applications`, which can be used for adding packages and applications. Any resources that are added in this location will only be available to web*Mathematica*.

$BaseDirectory

The directory, `$BaseDirectory`, is provided to install resources such as packages and applications so that they are available to all users of *Mathematica* and all installations of *Mathematica* on a machine. Packages and applications can be installed in `$BaseDirectory/Applications` where they will be available to web*Mathematica*.

$UserBaseDirectory

The directory, `$UserBaseDirectory`, is provided to install resources such as packages and applications so that they are available to a particular user of *Mathematica* on a machine. Packages and applications can be installed in `$UserBaseDirectory/Applications` where they will be available only to the particular user who is running the web*Mathematica* server.

The Script Directory

Another location for packages and applications is in the directory in which the JSP script lives. By default files cannot be loaded from this directory. It is possible to add the location to the *Mathematica* path using `MSPPageDirectory`. This is demonstrated below, using `Needs` to load a package and `Get` to load a data file.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<msp:evaluate>
    Block[{$Path=Append[$Path, MSPPageDirectory[]]},
        Needs["MyPackage`"];
        Get["Data.m"];
    ]
</msp:evaluate>
```

This arrangement would allow the directory of JSPs and code to be moved to another installation of web*Mathematica* with a minimum of rearrangement.

A drawback to this technique is that the code, `MyFile.m`, is available to be downloaded directly from the web server by a direct request. If it contained private information, this might be considered a security risk.

$TopDirectory

It is possible to install packages and applications inside the *Mathematica* layout. This makes them only available to that particular installation of *Mathematica*. Generally this is not recommended.

PackagesDirectory

It is possible to put packages and applications into a directory specified by the PackagesDirectory configuration parameter.

Absolute Filename

A last way of installing code is to use an absolute pathname. An example is the following.

```
<msp:evaluate>Get[ "d:\\My
Work\\LastOneThatWorked\\MyFile.m"]</msp:evaluate>
```

This type of loading is very common and is nearly always a very bad idea. It leads to fragile code that requires a significant amount of maintenance. For very little extra effort one of the previous methods should be used.

5.8 Alternative Server Technologies

There are various different server technologies that can be used in conjunction with web*Mathematica*. One basic strategy is to use URLs that refer to computations for a web*Mathematica* server, a technique that should work for any server technology. This is particularly easy for `img` tags.

Here is an `img` tag that refers to a computation carried out in web*Mathematica*; it could be embedded in the result from any server technology.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<img
    src="http://myserver:8080/webMathematica/MSP/test?arg1=val1&arg2=v
    al2"/>
```

Tighter cooperation between a specific server technology and web*Mathematica* is often possible. Remember that web*Mathematica* is based on servlets so any question about interoperation of a server technology and web*Mathematica* is really a question of interoperation of the server technology and Java Servlets and JavaServer Pages. Some specific examples of integration follow.

5.8.1 JavaServer Pages

JavaServer Pages (JSP) technology is an extension of Java Servlet technology. It is the Java equivalent of MSP technology allowing Java code to be embedded in HTML pages. Since web*Mathematica* has a JSP implementation, it is very strongly integrated with JSPs.

5.8.2 PHP

PHP is a server-side, cross-platform, HTML-embedded scripting language (http://www.php.net). There is a PHP extension that allows interaction of PHP and servlets, which is available at http://cvs.php.net/cvs.php/php4.fubar/sapi/servlet . This link contains a README file, which is a good place to start to integrate PHP and web*Mathematica*.

5.9 PDF Documents

One of the important uses of *Mathematica* is as an electronic format for technical documents. This derives in large part from the properties of *Mathematica* notebooks, which provide a document format that combines text, mathematics, and graphics suitable for many technical areas. It is quite possible to use a web*Mathematica* site to generate technical documents. One format that is very easy to generate is *Mathematica* notebooks, and there are a number of web*Mathematica* examples that work by returning *Mathematica* notebooks to the client. One example is Content.jsp. Using *Mathematica* notebooks has the advantage that the document can be worked on further after reaching the client. A disadvantage is that it requires the client to have access to an application that can read notebooks, such as *Mathematica* or *MathReader* (http://www.wolfram.com/products/mathreader/). Another alternative is to use PDF. This has the advantage that the vast majority of clients are set up to render PDF. This section will explore how to generate PDF documents from web*Mathematica*.

Mathematica, currently, does not have built-in technology for generating PDF, so external tools are required. The steps for generating PDF involve generating a *Mathematica* notebook, converting the notebook to PostScript, and converting the PostScript to PDF. These steps are discussed in the following sections.

Generating a *Mathematica* Notebook

A common way to do this involves using *Mathematica* commands for generating notebooks. A sample function is shown below (taken from the source used by Content.jsp).

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
MakeNotebook[] :=
    Developer`UseFrontEnd[
        Module[ {nb, nbobj},
            nb = NotebookCreate[] ;
            NotebookWrite[ nb, Cell[ "A Dynamically Created
    Notebook", "Title"]] ;
            NotebookWrite[ nb,
                Cell[ "Converted to " <> $$button, "Subtitle"]] ;
            NotebookWrite[ nb, Cell[ "The date is " <> ToString[
    Date[]], "Text"]] ;
            nbobj = NotebookGet[ nb] ;
            NotebookClose[ nb] ;
            nbobj]]
```

This sample shows how a notebook object is created with `NotebookCreate`, and then how the content is added with `NotebookWrite`. When the notebook is complete, a *Mathematica* expression holding the notebook is obtained with `NotebookGet` and this is returned. In a real life situation, the document could contain graphics, more text, and some computations. The *Mathematica* documentation has much more information on the commands for generating notebooks.

A major advantage of working with *Mathematica* notebooks like this is that they will take care of details such as font selection, graphics, and mathematics without the author having to be very involved.

Converting to PostScript

This is done quite easily with the function `NotebookPrint`. The following writes a PostScript representation of the notebook object into `file.ps`. The notebook object could be generated with the techniques in the previous section.

```
NotebookPrint[ nb, file.ps]
```

Converting PostScript to PDF

There are a number of tools for converting PostScript to PDF. Two that we have experience with are `ps2pdf` (http://www.cs.wisc.edu/~ghost/doc/AFPL/8.00/Ps2pdf.htm) and `PStill` (http://www.pstill.com). These are third-party tools that are not supported by Wolfram Research, but seem to work very satisfactorily. For both of them, it is necessary to make sure they can access all the special fonts that *Mathematica* uses. The documentation for the particular converter should be studied to see how to add extra fonts, which are available in the *Mathematica* layout.

©1988-2006 Wolfram Research, Inc. All rights reserved.

5.9.1 Creating PDF Example

This section describes an example that generates a notebook and converts it to PDF. For it to work you need to install PDF tools described in the last section. If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/PDF/Generate.jsp . (You may have some other URL for accessing your server.) The source is in `webMathematica/Examples/PDF/Generate.jsp` and `webMathematica/WEB-INF/Applications/ExampleUtilities/PDF.m`.

First, here is the JSP source.

```
<form action="Generate.jsp" method="post">
This example generates a notebook, converts it to PDF and returns
the PDF.
</p>

<msp:allocateKernel>

<msp:evaluate>
    If[ MSPValueQ[ $$button],
        Get[ "ExampleUtilities`Content`"] ;
        Get[ "ExampleUtilities`PDF`"] ;
        nb = MakeNotebook[] ;
        pdf = NotebookToPDF[ nb] ;
        If[ StringQ[ pdf],
            MSPReturn[ pdf, "application/pdf"],
            "Conversion failed, perhaps the PDF tools are not
     installed."]]
</msp:evaluate>

</msp:allocateKernel>

<br>
<br>
<input type="submit" name="button" value="Generate">

</form>
```

This code loads a couple of packages for creating notebooks and converting to PDF. It calls the function `MakeNotebook`, which generates a very simple notebook. It then passes the notebook object that was created into the function `NotebookToPDF`. If this returns a string, it is returned to the client using `MSPReturn`.

Here is the *Mathematica* source for the function `NotebookToPDF`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
$PStillCommand = "/usr/local/pstill_dist/pstill"

PDFCommand[ out_, in_] :=
    $PStillCommand <> " -o " <> out <> "  " <> in

NotebookToPDF[nb_]:=
    Module[ {stm1,stm2,file1,file2,res},
        stm1= OpenTemporary[];
        stm2=OpenTemporary[];
        Close[stm1];
        Close[stm2];
        file1=First[stm1];
        file2=First[stm2];
        Developer`UseFrontEnd[ NotebookPrint[nb, file1]];
        Run[ PDFCommand[ file2, file1]];
        res=Read[file2, Record,RecordSeparators -> {}];
        DeleteFile[file1];
        DeleteFile[file2];
        res
        ]
```

Note that this requires a definition to point to the location of the conversion tools. In this case it is set to use `PStill`, but it could easily be modified to use `ps2pdf`. Note also that the command is suitable for a Unix file system. On Windows it could be set as shown below; note the doublequotes around the command and the doubling of backslashes.

```
$PStillCommand="\"C:\\Program Files\\GPStill\\pstill.exe\""
```

The code first creates some temporary files and then sends PostScript to one of the files with the command `NotebookPrint`. It then uses the `Run` command to invoke the PDF conversion on the PostScript, and finally reads the PDF into a *Mathematica* string and returns this as the result.

There are all sorts of further extensions to these simple functions that interface with the PostScript to PDF conversion tools. For example, they could be extended to pass various options to the converter.

5.10 Returning General Content

The typical result of a web*Mathematica* request is an HTML page, which might include references to images. The commands available for web*Mathematica* are designed to make this type of request very convenient. However, it is also very useful to be able to return other formats such as *Mathematica* notebooks or TeX documents. *Mathematica* commands for generating these other formats are `Export` and `ExportString`. When these other formats are returned to a browser, it can often launch a helper application that provides special functionality for that format. This section discusses how to use web*Mathematica* to return general content of different formats. The functionality is based on two commands: `MSPReturn` and `MSPURLStore`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

5.10.1 MSPReturn

When an MSP script evaluates MSPReturn, the processing of the script is terminated and the first argument is immediately returned. The second argument specifies the content type. In this example a notebook object is returned, and the result is set to be application/mathematica.

```
MSPReturn[
    "Notebook[Cell[\"Hello\",\"Title\"]]","application/mathematica"]
```

Certain HTTP clients can use the content type to launch a helper application. However, some clients need a filename to be associated with the request. For this purpose MSPReturn takes a third argument that sets the filename in an HTTP header.

```
MSPReturn[ "Notebook[Cell[\"Hello\",\"Title\"]]",
        "application/mathematica","notebook.nb"]
```

However, for some HTTP clients (for example Internet Explorer) this has the undesirable effect of causing the client to display two Open or Save dialog boxes. Most clients work much better if the request for the script that contains the MSPReturn uses the filename with an appropriate extension. Since the extension for web*Mathematica* requests has to end in .jsp, this is not possible. An alternative is to generate a URL that has the correct extension; this functionality is provided by MSPURLStore.

5.10.2 MSPURLStore

MSPURLStore uses the mechanism that web*Mathematica* provides for storing images generated by commands such as MSPShow. It actually stores its argument on the server and returns a URL that references the argument.

```
In[1]:= Needs["MSP`"]

In[2]:= m = DisplayString[Graphics[Line[ {{0, 0}, {1, 1}}]], "JPEG"];

In[3]:= MSPURLStore[m, "image/jpeg"]

Out[3]= /webMathematica/MSP?MSPStoreID=FileNameBase_186159533&MSPStoreType=image/jpeg
```

The URL is relative to the request that contained the MSPURLStore. It contains a unique identifier and a description of the content type. Since the server steadily deletes stored information on the server, the information will not remain on the server indefinitely. This mechanism is particularly useful for preparing input for plug-ins and applets.

MSPURLStore can also take a third argument to set the filename, which is put into the URL that is returned. For example, the filename of notebook.nb is set in this example.

```
In[4]:= MSPURLStore[ "Notebook[Cell[\"Hello\",\"Title\"]]",
         "application/mathematica", "notebook.nb"]
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
Out[4]=  /webMathematica/MSP/notebook.nb?MSPStoreID=
           FileNameBase_682425268&MSPStoreType=application/mathematica
```

The example script `Examples/ContentStore.jsp` has an example of `MSPURLStore`.

5.10.3 MSPPageOptions/ContentType

The `ContentType` option of `MSPPageOptions` offers a way to set the content type of a particular page. Generally this would be placed at the top of a page. The following example shows how a page can set its return type to be MathML.

```
<msp:evaluate>
    MSPPageOptions[ ContentType -> "text/mathml"]
</msp:evaluate>

<msp:evaluate>
    MSPFormat[ Integrate[ 1/(1-x^3),x], StandardForm, RawMathML]
</msp:evaluate>
```

This is, of course, quite similar to `MSPReturn`. It is different in that it returns the entire page, whereas `MSPReturn` returns only its first argument.

©1988-2006 Wolfram Research, Inc. All rights reserved.

6. Advanced Topics

This section discusses a variety of more complicated and advanced details of a web*Mathematica* site.

6.1 Variables

This section discusses the use of variables in web*Mathematica* pages. It covers the way that input variables are processed as well as issues, such as scoping, that concern the use of local variables in code that appears in web*Mathematica* pages.

There are three kinds of variables in web*Mathematica*: input variables, page variables, and session variables. Variables that start with `$$` are input variables; they are given values from the HTTP request and are cleared when the kernel is cleaned when the page is finished. Variables that do not start with a $$ only get values if they are assigned values. These assignments last until the page is cleared if they are page variables, and for the lifetime of an HTTP session if they are session variables.

The details of these variables are now discussed.

6.1.1 Input Variables

Input variables are named to start with `$$` and are given values if their names are sent with the HTTP request. They have a special name because it is important to know which are the input variables. In the example below, the input variable `$$setting` will get the value entered into the `input` element (because the `input` element uses the name `setting`). You can test if a variable has a value with the MSP function `MSPValueQ`.

```
<input type="text" name="setting" />

<msp:evaluate>
    If[ MSPValueQ[ $$setting],
        ....
        ]
</msp:evaluate>
```

One important decision governs whether the value of input variables should or should not be interpreted. If the actual string value of the variable is suitable for your uses, you should use it. Alternatively, the string value may represent some input to a *Mathematica* command and it will be necessary to interpret it. For interpretation it must be something that *Mathematica* can interpret and the result must pass validation by the security system. If you find that you are starting to modify the security system, you should consider working with the uninterpreted values.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Interpretation of Input Variables

If you want *Mathematica* to compute with an input variable, it must be interpreted. web*Mathematica* provides various functions to help interpretation. It is important that you use these functions because they make use of the security features. If you try to bypass them, you could compromise the security of your system.

`MSPBlock` and `MSPToExpression` are provided to obtain expressions from input variables. This is, of course, completely essential for any type of interactivity. There are two stages to this process: a first stage involves interpretation and a second stage involves validation. Interpretation determines the input to *Mathematica*, and validation ensures that the *Mathematica* commands to be executed do not endanger the security of your site.

Interpretation is based on the *Mathematica* function `ToExpression` that calls a parser to try to determine input to *Mathematica*. Valid input for `MSPBlock` and `MSPToExpression` can either be regular *Mathematica* input, as documented in *The Mathematica Book*, or MathML. The following examples demonstrate input processing with `MSPToExpression`, showing both *Mathematica* and MathML input.

First, load the MSP application and then lock down the security system. More information is found in the section on Security.

```
In[1]:= Needs[ "MSP`"]

In[2]:= SetSecurity[];
```

Now you can use `MSPToExpression` to interpret some *Mathematica* input.

```
In[3]:= $$e1 = "x^y"

Out[3]= x^y

In[4]:= MSPToExpression[ $$e1]

Out[4]= x^y
```

Here the input is MathML.

```
In[5]:= $$e2 = "<math><msup><mi>x</mi><mn>2</mn></msup></math>"

Out[5]= <math><msup><mi>x</mi><mn>2</mn></msup></math>

In[6]:= MSPToExpression[ $$e2]

Out[6]= x^2
```

`MSPBlock` provides additional functionality to work with the interpreted value of an input variable. This is described further in the *Mathematica* Function Reference section.

web*Mathematica* carries out validation to ensure the security of the system. Validation involves checking any input that was sent to the server to see if it is safe to be used in a *Mathematica* computation. You can find more information on the topic in the Security section.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Interpreted versus Noninterpreted Values

As described above, whenever you work with an input variable, you need to decide how to work with its value. You can work with the noninterpreted value and make choices based upon its setting. (This will be a string.) Alternatively, you can interpret the value so that it can be used for computation in *Mathematica*. This section gives an example of working with both interpreted and noninterpreted values. If you installed web*Mathematica* as described above, you should be able to connect to this JSP via http://localhost:8080/webMathematica/Examples/-SimplifyIntegrate.jsp.

Here is a listing of the `form` element from the JSP `SimplifyIntegrate.jsp`.

```
<form action="SimplifyIntegrate.jsp" method="post">
<msp:allocateKernel>
Input:
<br>
<msp:evaluate>
    integrand = Null;
    If[ MSPValueQ[ $$expr],
        integrand = MSPToExpression[ $$expr]] ;
</msp:evaluate>

<input type="text" name="expr"  size="24"
          value="<msp:evaluate> MSPValue[ $$expr,
    "Sin[x]^2"]</msp:evaluate>" />
<br/>
<br/>
<msp:evaluate>
    If[ integrand =!= Null,
        res = Integrate[ integrand,x] ;
        If[ $$simplify === "on", res = Simplify[ res]] ;
        MSPFormat[res,StandardForm]]
</msp:evaluate>

<br/>
<input type="submit" name="btnSubmit" value="Evaluate">
<br/>
Simplify result:
<input type="checkbox" name="simplify"
    <msp:evaluate> If[ $$simplify === "on",
    "checked=\"checked\""]</msp:evaluate>>
<br>
</msp:allocateKernel>
</form>
```

In this example, there are two input variables, `simplify` and `expr`, that may be submitted with a request. The first of these is not interpreted and is used to select the action to be taken. Only the second is actually interpreted as *Mathematica* input. This is manifest in the contents where `$$expr` is interpreted with `MSPToExpression`. This is necessary because it represents a general *Mathematica* expression and will be used as input to *Mathematica*'s `Integrate` function. The setting of the check box is carried by the variable `$$simplify`, and this is tested to see if it has the value `"on"`. There is thus no need to interpret it. In general it is better not to interpret if it can be avoided.

©1988-2006 Wolfram Research, Inc. All rights reserved.

MSPBlock versus MSPToExpression

web*Mathematica* provides two MSP functions for interpreting input variables: `MSPBlock` and `MSPToExpression`. This section contrasts and compares them.

`MSPBlock` is probably the simplest of the two. It offers a compact and simple way to interpret and use the value of an input variable, as shown below.

```
<msp:evaluate>
    MSPBlock[ {$$expr, $$num},
        Expand[ $$expr^$$num]]
</msp:evaluate>
```

Remember that the `$$expr` in the body, used here in an `Expand` computation, refers to the interpreted value of the input variable `$$expr`. If the value of `$$expr` cannot be interpreted or fails a security test, an exception will be thrown. If `$$expr` has no value, then the `MSPBlock` will not be evaluated, instead returning a null result.

An alternative way to interpret input is to use `MSPToExpression`. This can use page variables to hold the result of interpretation. It is not quite as neat as the use of `MSPBlock`, but is somewhat more expressive. Here is an example.

```
<msp:evaluate>
    poly = Null;
    exponent = Null;
    If[ MSPValueQ[ $$expr, $$num],
        poly = MSPToExpression[ $$expr] ;
        exponent = MSPToExpression[ $$num]] ;
</msp:evaluate>

<msp:evaluate>
    If[ poly =!= Null && exponent =!= Null,
        Expand[ poly^exponent]]
</msp:evaluate>
```

This example shows how web*Mathematica* extracts the interpreted value of `$$expr` and stores it with the page variable `poly`. This is especially useful if you use the interpreted value in a number of different places.

6.1.2 Page Variables

Standard *Mathematica* programming constructs such as `Block`, `Module`, and `Function` all work in their typical ways with regard to localization and scoping issues. You can find more information on their operation in standard *Mathematica* references.

©1988-2006 Wolfram Research, Inc. All rights reserved.

You can use variables in *Mathematica* code inside of `msp:evaluate` tags; they can store intermediate values and be used for computation. Since these variables will be cleared when the kernel is cleared, it is not important to put these variables into a normal *Mathematica* program structure such as `Module` or `Block`. Consequently these are called page variables.

In the example below the page variable `tmp` holds the value `Null`; it then gets the result of calling `MSPToExpression` on the input variable `$$expr`. If `MSPToExpression` was unable to complete its task, for example, because of a security error, it will throw an exception and `tmp` will still have the value `Null`. Later, if `tmp` is not set to `Null`, you can be certain that the input was processed without problems and you can use it for calculations.

```
<msp:evaluate>
    tmp = Null;
    tmp = MSPToExpression[ $$expr] ;
</msp:evaluate>

<p>
<msp:evaluate>
    If[ tmp =!= Null,
        ....
        ]
</msp:evaluate>
</p>
```

When the page is finished, `tmp` will be cleared.

6.1.3 Session Variables

If you want to save any values from one request to the next, you can use `MSPSessionVariable` to make a session variable. These will be stored in the server and can be used in pages that are part of different requests. They use HTTP sessions and so the session variables for one user are not visible to those of another user (just as the shopping cart at an e-commerce site is for one user and is not visible to another).

In the code fragment below there are two variables, `savedInput` is a session variable, declared with `MSPSessionVariable`, while `xInput` is a page variable. In the second evaluation, if `xInput` has a value, this is added to `savedInput`.

```
<msp:evaluate>
    MSPSessionVariable[ savedInput, {}];
    xInput = Null;
    xInput = MSPToExpression[ $$expr] ;
</msp:evaluate>

<p>
<msp:evaluate>
    If[ xInput =!= Null, savedInput = Append[ savedInput, xInput]];
</msp:evaluate>
</p>
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

You can work with session variables in just the same way that you work with page variables; assigning them the results of calculations and then later retrieving them. The difference is that session variables last after the page is finished.

An example of `MSPSessionVariable` is shown in Session.jsp.

6.2 Security

Internet security concerns both client and server security. However, web*Mathematica* is a server technology. Consequently, there are no special client security issues relating to web*Mathematica* JSPs. Of course, you still need to consider client-side security, but *Mathematica* does not introduce any special client security issues. The real issue is the security of the server.

IMPORTANT SUMMARY: If you do not have time to read all of this security section or you are unsure about some of the terms, try to remember one important rule about security. NEVER use `ToExpression`; instead always use the secure function `MSPToExpression`.

Running a general computation system like *Mathematica* inside of a web site presents many potential security hazards for the server. *Mathematica* contains commands for examining and deleting files and for launching arbitrary processes. A number of security features are built into the web*Mathematica* tools, but these are designed to work in conjunction with other standard security features. If security is important to you, study and use features such as firewalls, trusted hosts, and HTTP-specific security features.

A major danger to any *Mathematica* site is that someone will try to send commands to *Mathematica* that may breach the security of the server. These commands can be sent as the value of input variables passed in from the server. The previous section on Interpretation of Input Variables discussed how input is interpreted by the system. This section concentrates on the validation process. It is possible for the system to receive input such as:

```
ReadList["/etc/passwd"]
```

or

```
Run["telnetd -d"]
```

Even if used as an argument to a seemingly harmless *Mathematica* command, they still pose an attack.

Try using one of these as an input in an example, such as http://localhost:8080/webMathematica/Examples/-Expand.jsp.

This type of input fails because input values are sent to *Mathematica* by the server as strings. In the *Mathematica* kernel, they should be processed by special functions, such as `MSPBlock` or `MSPToExpression`, which provide a secure way to interpret input. In addition, a security feature is built into the *Mathematica* command `ToExpression` that prevents inadvertent calls that bypass the security system.

©1988-2006 Wolfram Research, Inc. All rights reserved.

6.2.1 MSP Function Validation

The MSP functions MSPBlock and MSPToExpression are provided to work with input to a web*Mathematica* site in a secure way. These functions parse the expression but validate it before allowing it to evaluate. If validation succeeds, the input can be used for further computation. If it is not validated, an MSPException is thrown. Here is an example of how this works.

```
In[1]:= Needs["MSP`"]

In[2]:= SetSecurity[];

In[3]:= $$e = "ReadList[\"/etc/passwd\"]";

In[4]:= MSPToExpression[$$e]

        Throw::nocatch :
         Uncaught Throw[{$$e, ReadList["/etc/passwd"]}, MSPException[S … or]]
           returned to top level.

Out[4]= Hold[Throw[{$$e, ReadList["/etc/passwd"]}, MSPException[SecurityError]]]
```

The message about an uncaught Throw arises because there is no Catch statement. A default handler will catch any exceptions that are raised when a page is processed; typically, it inserts some suitable text. It is also possible for an author to catch these exceptions and issue some special error message.

Important: Security Warning

If you want your pages to be secure from attack, it is important that you understand what your pages are computing. In particular, commands that open, close, or carry out other operations on files should be dealt with carefully, as should any commands, such as Run or LinkOpen, that start processes. If you use ToExpression, exercise caution as described below.

The Validation Process

The validation process works in a straightforward manner, and you can customize it to give more or less security. You can investigate its operation in the following steps:

First, load the MSP *Mathematica* application and then lock down the security model, which cannot be modified after SetSecurity is called. When the server initializes *Mathematica*, it calls SetSecurity.

```
In[1]:= Needs["MSP`"]

In[2]:= SetSecurity[];
```

Now you can test expressions for validity. A first example shows a harmless mathematical expression that is found to be secure.

```
In[3]:= InsecureExprQ[ HoldComplete[ Sin[6]]]

Out[3]= False
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

Here is a less-than-friendly expression, the sort of thing that could be sent as an attack.

```
In[4]:= InsecureExprQ[ HoldComplete[ Run[ "telnetd"]]]

Out[4]= True
```

Validation works by collecting all the symbols into a list and steadily reducing the list. If any symbols remain after reduction, the expression is not secure. The reduction process works with lists of symbol and context names that can either be allowed or disallowed according to the following steps.

- If `AllowedContexts` is a list, remove symbols with contexts on this list.
- If `AllowedContexts` is not a list, remove symbols with contexts not in `DisallowedContexts`.
- If `AllowedSymbols` is a list, remove symbols found in this list.
- If `AllowedSymbols` is not a list, remove symbols that are not in `DisallowedSymbols`.
- If no symbols remain, the expression is secure; otherwise it is not secure.

These tests allow you to be restrictive or flexible. If you use the allowed lists, you are restrictive and have more security, whereas if you use the disallowed lists, you are less restrictive with less security. It is up to each individual site to decide the appropriate balance.

When the server is started, a default security model is installed. This default security model looks like this.

```
In[5]:= MSP`Utility`AllowedContexts

Out[5]= {Global`}

In[6]:= MSP`Utility`DisallowedContexts

Out[6]= MSP`Utility`DisallowedContexts

In[7]:= MSP`Utility`AllowedSymbols

Out[7]= HoldComplete[Plus, Times, Power, Sqrt, Log, Exp, HoldComplete, ∞, π, ℯ, °,
         GoldenRatio, Catalan, EulerGamma, OutputForm, StandardForm, List, Sin,
         Cos, Tan, Sec, Csc, Cot, Sinh, Cosh, Tanh, Sech, Csch, Coth, ArcSin, ArcCos,
         ArcTan, ArcSec, ArcCsc, ArcCot, ArcSinh, ArcCosh, ArcTanh, ArcSech, ArcCsch,
         ArcCoth, True, False, Derivative, D, Dt, ⅈ, Greater, Less, GreaterEqual,
         LessEqual, Inequality, Equal, Re, Im, Abs, Sign, Conjugate, Arg, Round, Floor,
         Ceiling, Max, Min, Mod, Quotient, Not, And, Or, Xor, Union, Intersection,
         Complement, AiryAi, AiryAiPrime, AiryBi, AiryBiPrime, BesselJ, BesselK,
         BesselI, BesselY, Factorial, Binomial, Multinomial, Gamma, Beta, LogGamma,
         PolyGamma, LegendreP, SphericalHarmonicY, HermiteH, LaguerreL, Erf, Erfc,
         Erfi, InverseErf, InverseErfc, ClebschGordan, ThreeJSymbol, SixJSymbol, Zeta,
         FresnelS, FresnelC, CosIntegral, SinIntegral, ExpIntegralE, ExpIntegralEi,
         SinhIntegral, CoshIntegral, HypergeometricPFQ, Hypergeometric0F1,
         Hypergeometric1F1, Hypergeometric2F1, HypergeometricPFQRegularized,
         MeijerG, AppellF1, EllipticK, EllipticF, EllipticE, EllipticPi, JacobiZeta,
         EllipticNomeQ, EllipticLog, InverseEllipticNomeQ, JacobiAmplitude, EllipticExp,
         DiracDelta, UnitStep, DiscreteDelta, KroneckerDelta, Identity, Function,
         Slot, GrayLevel, Hue, RGBColor, CMYKColor, Automatic, None, All, Null, O, C]

In[8]:= MSP`Utility`DisallowedSymbols
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
Out[8]= MSP`Utility`DisallowedSymbols
```

This model will allow any symbol in `Global`` context, in addition to a number of specific symbols. This is a fairly restrictive model that provides a higher level of security.

6.2.2 Setting Your Own Security Model

You can make your own definitions for `MSP`Utility`AllowedContexts`, `MSP`Utility`DisallowedContexts`, `MSP`Utility`AllowedSymbols`, and `|`. These definitions should be placed into a file in the `/WEB-INF/conf` directory and the name of the file set by the configuration parameter `SecurityConfigurationFile`. For example, if the configuration information is in a file called `SecurityConfiguration.m`, the following should be added to MSP.conf.

```
SecurityConfigurationFile=SecurityConfiguration.m
```

A sample security configuration file is shown below. This only allows symbols in the `Global`` context in addition to `Plus`, `Times`, and `Power`. This is a particularly restrictive security system that might be appropriate in some circumstances.

```
MSP`Utility`AllowedSymbols =
    HoldComplete[ Plus, Times, Power]

MSP`Utility`AllowedContexts =
    {"Global`"}
```

As described in the section on Multiple Kernel Pools, it is possible to use different configuration details for different request URLs. Each pool has its own configuration file and its own security system.

When each *Mathematica* kernel is launched, these four security parameters are sent to the log system.

You can test your security model from within *Mathematica*. You first need to have installed the MSP *Mathematica* application into your *Mathematica* layout. This installation is not necessary to run web*Mathematica*, but it is necessary if you want to use the functions in an interactive session of *Mathematica*. It was described earlier.

You can then run the functions from within a *Mathematica* session. First, you need to load the MSP application.

```
In[1]:= Needs["MSP`"]
```

Now you need to place your security configuration into a file and then load this into *Mathematica*. Here the example has used the configuration information shown above and the file is placed into the `C:\Temp` directory.

```
In[2]:= SetSecurity["C:\\Temp", "MySecurity.m"]

Out[2]= True
```

This input passes the security model since it contains symbols in the `Global`` context and `Plus`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
In[3]:= MSPToExpression[ "x+y"]

Out[3]= x + y
```

However, this input does not pass the security system since it also contains the Sin symbol which is not permitted by the security system.

```
In[4]:= MSPToExpression[ "Sin[x]+y"]

        Throw::nocatch :
         Uncaught Short[Throw[{Sin[x]+y, Sin[x]+y}, MSPException[SecurityError]]]
           returned to top level.

Out[4]= Hold[Throw[{Sin[x]+y, Sin[x]+y}, MSPException[SecurityError]]]
```

When you have determined the appropriate security system, you can place your security configuration file into the webMathematica/WEB-INF/conf directory and set the SecurityConfigurationFile parameter of MSP.conf.

6.2.3 ToExpression Validation

It is possible that someone developing MSP scripts may not fully understand the security system. They may decide to call ToExpression directly on an input variable, which would bypass the security system and pose a potential danger. To provide security for such events, ToExpression is validated if it has a first argument equal to any of the inputs sent with the request. In this way, the security system will validate a computation such as the following.

```
<msp:evaluate>
    val = ToExpression[ $$num]
</msp:evaluate>
```

If the value of $$num passes the security test, ToExpression works as expected. However, if the security test fails, a security exception will be thrown.

You can disable this security test by setting the *Mathematica* variable MSP`Utility`CheckToExpression to False. In addition, you can disable the test in MSP.conf.

```
CheckToExpression=false
```

It is probably an exceptional site that disables this security feature.

Of course, if the string input to ToExpression comes from an input sent with the request, but is modified in some way, the call to ToExpression will not carry out any validation. Due to this, it is highly recommended that you never use ToExpression, but instead use MSPToExpression.

©1988-2006 Wolfram Research, Inc. All rights reserved.

6.2.4 Access Restrictions

You may wish to restrict access to certain parts of your system such as the kernel monitor, which is provided for monitoring and debugging your system. In this case, refer to the sections on Logging and the Kernel Monitor. The section on Apache and Tomcat describes how this can be when web*Mathematica* is used from the Apache web server.

6.3 Evaluation Formatting

The output of an `msp:evaluate` tag is inserted into the page that is returned as part of the web request. This section will describe the different types of formatting output. This topic is related to the placement of *Mathematica* commands into web*Mathematica* pages and more information is found in `Appendix: <msp:evaluate>`.

Automatic Formatting

Any result that is computed by an `msp:evaluate` tag that is not a string will be formatted into a string that will use the necessary HTML escapes. An example is shown below.

```
<msp:evaluate>
    Range[5]
</msp:evaluate>
```

This type of formatting is equivalent to `MSPFormat` with a format type of `OutputForm`. The reference section has more information on `MSPFormat`.

MSPFormat

Different styles of formatting output can be generated with `MSPFormat`. The example below uses `MSPFormat` with a format type of `TraditionalForm`.

```
<msp:evaluate>
    MSPFormat[ Sqrt[ Sin[x]], TraditionalForm]
</msp:evaluate>
```

Output can be generated formatted into HTML, MathML, and images. The latter gives a convenient way to show typeset mathematics. The reference section has more information on `MSPFormat`.

String Formatting

If the result of `msp:evaluate` is a string, it is left unmodified and added to the output page. This is often useful for constructing HTML, as shown in the example below.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<msp:evaluate>
    StringJoin[ "<b>", ToString[ x], "</b>"]
</msp:evaluate>
```

If you have a string and you want it to be formatted with HTML escapes, then you can wrap it in `MSPFormat`.

Graphics and Image Formatting

There are several convenient functions for formatting graphics objects so that a picture appears in the output. The example below displays a plot.

```
<msp:evaluate>
    MSPShow[ Plot[Sin[x],{x,0,2Pi}]]
</msp:evaluate>
```

The reference section has more information on `MSPShow`.

Suppressing Output

Sometimes you want to use the `msp:evaluate` tag to evaluate something, but you do not want to leave anything in the resulting page. This can be done by adding a semicolon ';' after the computation as shown below.

```
<msp:evaluate>
    Needs[ "MyPackage"];
</msp:evaluate>
```

Adding a semicolon causes the *Mathematica* symbol `Null` to be returned, and this is formatted to leave no trace in the output.

Output is suppressed whatever the computation, whether it uses one of the formatting functions, a graphics function, or a function that returns `Print` or `Message` output. In the following example, no output will be seen from the message output function because it is followed by a semicolon.

```
<msp:evaluate>
    MSPGetMessages[];
</msp:evaluate>
```

Multiple Calculations

If you wish to calculate more than one result in an `msp:evaluate` tag, the different steps must be separated with a semicolon ';'. The result of the last computation will be formatted and appear in the output page. In the example below, the numerical result of `x+y` computation will appear.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<msp:evaluate>
    x = Sin[5.6];
    y = Sqrt[x];
    x+y
</msp:evaluate>
```

If you wish to suppress the result of the last computation, you can use a semicolon ';' as described in the section on Suppressing Output.

6.4 Processing a JSP

This section will describe the different stages that are involved in processing a JSP for web*Mathematica*.

A JSP is processed as part of an HTTP transaction. A client sends a request to the server, which replies with a response. One feature of HTTP requests is that they can send parameters and values to the server. This is essential for any dynamic behavior, because parameters are used to select and control the response. The response could be an HTML page. However, it could be some other content type, such as an image, a *Mathematica* notebook, or some form of XML.

The JSP is processed by the servlet container in which it is running, it is processed in a top-down method, so that commands at the top are evaluated before commands lower down. A JSP interacts with web*Mathematica* by means of the custom tags defined in the MSP Taglib. A sample page is shown below.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>
<html>
<title>page</title>
<body>
<msp:allocateKernel>
    <msp:evaluate>
        eval1
    </msp:evaluate>
    <msp:evaluate>
      eval2
    </msp:evaluate>
</msp:allocateKernel>
</body>
</html>
```

The <msp:allocateKernel> tag takes responsibility for allocating a *Mathematica* kernel to use for computations, assigning input variables and parameters, and other initialization. Note that the kernel that was allocated will be available in a clean state. A later section describes the details of <msp:allocateKernel>.

Any <msp:evaluate> tags then use the allocated *Mathematica* kernel to evaluate their input. Note that any assignments or definitions made in one <msp:evaluate> tag will be visible in another. A later section describes the details of <msp:evaluate>.

Finally the </msp:allocateKernel> tag takes responsibility for releasing the *Mathematica* kernel, first cleaning it of any definitions that were made. If any special processing, for example to deal with exceptions, is

©1988-2006 Wolfram Research, Inc. All rights reserved.

needed this will be carried out at this point. A later section describes the details of </msp:allocateKernel>.

It should be noted that there are a number of other special tags that can be used; these are detailed in the section on the MSP Taglib.

6.5 Multiple Kernel Pools

For some applications, it is useful to use several pools of *Mathematica* kernels to serve different requests. You can configure the kernels in each pool differently, perhaps with different timeout parameters or different initialization files. Another possibility would be to use a pool with one *Mathematica* kernel as a demonstration server and another pool with four *Mathematica* kernels to serve customer requests. This would ensure that the customers received priority. Another benefit of multiple pools is the guarantee that one set of computations is completely isolated from another. Disadvantages, however, are the additional administration and the need for extra web*Mathematica* licenses.

Whenever web*Mathematica* is launched, there is always one pool created, called the General pool. If you wish to configure for additional pools, add a KernelPools statement to the MSP.conf file, as shown below.

```
KernelPools=Examples,Test
```

This specifies two additional pools, called Examples and Test. The system will look for the configuration files Examples.conf and Test.conf in the same directory as MSP.conf. In these configuration files, you can modify many of the parameters set in the main file. Any values that are not set will take their values from MSP.conf. If the pool-specific configuration file is missing, all of the values will be taken from MSP.conf. The documentation on MSP.conf describes the configuration parameters that can be placed in these files.

If you wish to work with multiple pools, you need to have web*Mathematica* licenses for all the kernels you wish to run. The Kernel Monitor contains information on all of the pools, so this is a good place to confirm that a pool has been properly initialized. The servlet log files also contain information about each pool, as the section on Logging describes.

6.5.1 Mapping JSPs to Pools

The JSP custom tag msp:allocateKernel is used to allocate *Mathematica* kernels. The pool that it uses to obtain a kernel is determined by the available pools and the name of the JSP. This is set by the MSP.conf setting of PoolPath, which maps the directories of JSPs to pools. An example is shown below.

```
KernelPools=Examples,Test
PoolPath.Examples=ExampleDir
PoolPath.Test=TestDir
```

In this example there are two kernel pools, Examples and Test. JSPs that are found in or beneath the top-level directory, ExampleDir, will use the Examples pool and those in the directory TestDir will use the pool Test. Other JSPs will use the General pool. For example, http://host/webMathematica/Example

©1988-2006 Wolfram Research, Inc. All rights reserved.

Dir/Files/Work.jsp, which refers to the file webMathematica/ExampleDir/Files/Work.jsp, will use the Examples pool.

6.5.2 Multiple Web Applications

Yet another way to divide requests is to install multiple web applications. This might be beneficial if your server provides special configuration tools for web applications. For this you would repeat the installation process for the web*Mathematica* web application, giving each new installation a different name. All web applications run in a common instance of the Java Virtual Machine will share the same kernel monitor.

6.6 Mapping URLs onto JSPs

The way that web*Mathematica* maps URLs onto JSPs is very straightforward. The URL names a JSP that lives directly in the web*Mathematica* web application or in a subdirectory. Some examples are shown in this section.

In the first table, the MSP Script.jsp is located in the top of the web*Mathematica* web application directory, for example, /usr/local/tomcat/webapps/webMathematica. It can be accessed by the URL http://host/webMathematica/Script.jsp.

	Top Directory
Script Name	Script.jsp
Script Location	/usr/local/tomcat/webapps/webMathematica
URL	http://host/webMathematica/Script.jsp

In this second table, Script1.jsp is located in a subdirectory of the web*Mathematica* web application directory, for example, /usr/local/tomcat/webapps/webMathematica/Test. The URL http://host/webMathematica/Test/Script1.jsp will find this JSP.

	Subdirectory
Script Name	Script1.jsp
Script Location	/usr/local/tomcat/webapps/webMathematica/Test
URL	http://host/webMathematica/Test/Script1.jsp

Remember that you should not place JSPs inside the WEB-INF directory. If you do, they will not be accessible.

6.7 *Mathematica* Initialization

web*Mathematica* uses a Java package that manages pools or groups of *Mathematica* kernels. It obtains a kernel from a pool, uses it, and returns it to the pool when finished. The kernel manager is responsible for launching and initializing, shutting down kernels at prescribed intervals, as well as timing out computations that exceed a preset time constraint. After a kernel has been shut down, it will be restarted and reinitialized. This section will briefly describe the initialization process and how it can be specially configured.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Each *Mathematica* kernel is launched and initialized as follows.

- The `MSP` application is loaded.
- The `ImageDirectory` is set.
- The `MSPDirectory` is added to `$Path`.
- The time constraint parameter is set.
- The `SecurityConfiguration.m` file is loaded (if present).
- The security system is locked down.

The MSP.conf file can contain two optional lines for configuration. They appear below.

```
PackagesDirectory=/usr/local/myPackages
KernelInitialization=Get[ "MyInit.m"]
```

web*Mathematica* adds the `PackagesDirectory` to the *Mathematica* `$Path` before any initialization is done, so it could be used to load the MSP application. The `KernelInitialization` setting is passed to the *Mathematica* kernel for evaluation as a last step of initialization. It can contain extra commands for loading special packages.

6.8 Logging

An administrator needs to confirm correct operation of a server and track down problems as they occur.

One key place to search for information on problems is the log file. This is a file to which the servlet container will write log information generated by web*Mathematica*. Under Tomcat, a typical location for the log file is `<path-to-tomcat>/logs/localhost_log.YYYY-MM-DD.txt`, where the filename includes the date. Under JRun, it is `<path-to-jrun>/logs/default-event.log`. For other servlet containers you will need to study the relevant documentation. If the log file is empty, it may indicate that the user running the servlet container does not have permission to write to the log file directory. The Unix command `tail -f` can be useful for monitoring output from this file.

The log file records serious errors; if your system does not function correctly at startup time, it would be good to look here. For example, if the configuration file is not found or the kernel cannot be launched, this will be recorded in the log file. Later, if there is a serious error that requires shutting down a kernel, this is also recorded.

The `VerboseLogs` configuration parameter selects between two levels of output to the log file. If it is set to `false`, each page access causes one line of information to be sent to the log file. This provides a convenient way to confirm that your site is running correctly. If `VerboseLogs` is set to `true`, a considerable amount of extra information is sent to the log file. This includes any messages the *Mathematica* kernel generates.

After your site is running, it is possible to modify the verbosity of logging and make other changes via the monitor, which will be described in the Kernel Monitor section.

©1988-2006 Wolfram Research, Inc. All rights reserved.

6.9 The Kernel Monitor

The kernel monitor is a servlet that collects information on the running of your site. You should be able to find the monitor via the URL http://localhost:8080/webMathematica/KernelMonitor. (You may have some different URL for accessing your server.) Upon access, the monitor brings up a page showing the current status of web*Mathematica*, describing various parameters of the site, and giving status information for each kernel. If you look at this page, access some JSPs, and then look at the page again, you should see updates, such as a change in the number of times kernels have been accessed.

The section of the monitor that displays kernel status shows useful information that can help you study the performance of your site, including how much time each kernel has been running and has spent servicing requests. It also shows the ratio of the two. If the pool has more than one kernel, it also shows averages of all kernels. You can monitor the load on the site by studying the ratio of time spent servicing requests and use the information to tune its performance.

The page provides a number of controls: at the top are buttons that cause all kernels to restart, the monitor to reload, and any logs that have been collected in the servlet to be cleared. There is a button to turn verbose logging on and off and a button that includes the entire log file in the output of the monitor.

Finally, there are two buttons that allow the user to capture and then display input to *Mathematica* that can be used for debugging purposes. The section on Debugging MSPs describes the operation of these buttons.

For security purposes it would be sensible to restrict access to the kernel monitor. If the servlet engine is accessed via an Apache web server, access can be restricted in the server configuration files. The section on Apache and Tomcat describes how this can be done.

6.10 Debugging web*Mathematica*

web*Mathematica* involves running computations inside of a server. This poses a number of problems and constraints for debugging. In general, debugging anything running inside a server can be difficult.

If you are authoring web*Mathematica* pages and find that your material does not work as you expect, you can use messages and print statements to resolve your problems. You can get message output returned in your web page with `MSPGetMessages` and the output of print statements with `MSPGetPrintOutput`. The capture of message and print output is described in the example Messages.jsp. It is probably a good idea to confirm that your calculations work correctly in an interactive *Mathematica* session.

In addition to message and print output, you can use the logging and monitor features provided by the system. These are described in more detail in the sections on Logging and the Kernel Monitor. The simplest technique is to look at the files written by the servlet engine. A more sophisticated way is to use the monitor, which can be accessed via a URL, for example, http://localhost:8080/webMathematica/KernelMonitor. If you increase the level of log output by setting `VerboseLogs` to `true`, you will generate more output.

©1988-2006 Wolfram Research, Inc. All rights reserved.

6.11 Displaying Mathematics and Graphics

In order to display typeset mathematics and graphics, the server can generate images. These provide a simple way to view the results of computations. However, they suffer from the serious defect that they cannot be used by the client. They cannot be resized, drawn with different fonts, or viewed in some alternative way. It is also hard for a computer program to extract any meaning from an image. Alternatives to images exist. In the case of mathematics one can use MathML, for graphics one can use SVG, and an applet can be deployed for three-dimensional graphics. These alternatives are not always the appropriate solution and, for this reason, functions for generating images are provided.

When a *Mathematica* kernel generates an image, it is stored in a file on the server and adds a reference in the HTML file that is returned. For example, the following `img` element may be generated.

```
<img
     src="/webMathematica/MSP?MSPStoreID=MSPStore1042942578_0&amp;MSPSt
     oreType=image/gif" alt="Created by webMathematica"/>
```

The SRC attribute references the MSP servlet through a URL, which includes a parameter that gives the name of the file. The MSP servlet returns the contents of the file and periodically deletes old image files. The actual location in which image files are saved is a workspace directory provided by the servlet container.

More information about generating images can be found in the *Mathematica* Function Reference section, especially under `MSPFormat`, `MSPShow`, and `MSPShowAnimation`.

An alternative way to generate images is to use the *Mathematica* command `Export`, available for use with the function `MSPExportImage`. This provides more features, such as transparent backgrounds, but takes longer to generate. `MSPExportImage` always makes use of the *Mathematica* front end.

MSP Functions Returning Images

There are various MSP functions that return images: `MSPShow`, `MSPShowAnimation`, and `MSPExportImage`; in addition `MSPFormat` may return an image. The workings of these commands are detailed in the *Mathematica* Function Reference section. It should be noted that these all work by returning a string that contains the necessary `img` tag to reference the image file which is stored on the server. An example is shown below.

```
<msp:evaluate>
    MSPShow[ graphics]
</msp:evaluate>
```

Therefore, if the MSP function is followed by a semicolon ';', as shown below, this will suppress the output. The use of a semicolon to suppress output is discussed in the section on Evaluation Formatting.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<msp:evaluate>
    MSPShow[ graphics];
</msp:evaluate>
```

Another use of these functions is to embed their results into some other formatting function such as those in the HTML Package. The example below will return an HTML table with two images.

```
<msp:evaluate>
    Needs["MSP`HTML`"]
</msp:evaluate>

<msp:evaluate>
    HTMLTableForm[ {MSPShow[ g1], MSPShow[ g2]}]
</msp:evaluate>
```

6.12 Including Static Files

web*Mathematica* involves returning dynamically generated material. However, the web pages that web*Mathematica* generates may include static images, which might, for example, have been generated by a designer. web*Mathematica* comes with a number of images, such as banners and buttons, which you may use. This section discusses how web*Mathematica* pages can use static files. It will focus on image files, but the principles apply in general to other files.

Images are placed in HTML pages with an `img` tag; it is convenient for these tags to use a relative URL to refer to the server from which the HTML page originated. Web pages that use relative URLs are easy to move from one server to another. There are two types of relative URLs: those that start with a "/" character and those that do not. The following URL starts with a "/" character.

```
<img src="/webMathematica/images/banners/webm-white.gif" />
```

If web*Mathematica* returns an HTML page containing this URL, the browser will try to load the image from the `images/banners` directory within the web*Mathematica* web application. The servlet container is capable of returning this image, which should appear correctly. If these image requests do not work and you are using a servlet container as a backend to another web server, you should make certain that it forwards requests for images to the servlet engine.

An alternative is a relative URL that does not start with a "/" character. For example an HTML page, generated by a URL such as `http://server/webMathematica/Examples/DesignTemplates/HeatWave.jsp`, may contain an `img` tag such as the following.

```
<img src="heatwave/bullet.gif" />
```

In this case, the browser will try to retrieve the image with the URL `http://server/webMathematica/Examples/DesignTemplates/heatwave/bullet.gif`, which can be processed by the servlet container

©1988-2006 Wolfram Research, Inc. All rights reserved.

to return the appropriate image file. This is convenient because you can place images and JSPs together in the same directory.

©1988-2006 Wolfram Research, Inc. All rights reserved.

7. Troubleshooting

This section describes techniques for addressing problems and errors. You should first work through the Initial Checks section, which will help track down any general problems. If this does not help, study the Specific Problems section. Further information can be obtained from the web*Mathematica* area of the Wolfram Research support web site, http://support.wolfram.com/webmathematica. Finally, if you have not resolved your problem and are eligible for support you should look at the Reporting Problems section. If you are not eligible for support, you can use the web*Mathematica* forum, http://forums.wolfram.com/webmathematica/list/.

7.1 Initial Checks

If you try to use a browser to connect to a web*Mathematica* server, and it does not operate in the expected way, try the following steps.

7.1.1 Check the Server

Make absolutely certain that your servlet container is working correctly. If you cannot connect to the demonstration examples that come with your container, then web*Mathematica* is unlikely to work. Furthermore, Wolfram Research will not be able to give more than minimal assistance until your servlet container is working.

7.1.2 Check the URL

If your servlet container works but web*Mathematica* does not, make sure you are using a correct URL. These are case sensitive, so make sure you use capital letters in the same places as the documentation describes. For example, http://localhost:8080/webMathematica/Examples/Hello.jsp is the appropriate usage. Make sure that this URL is consistent with the way you connect to your servlet container. If you need to specify a port number for your servlet container, you will need to use this for web*Mathematica*. For example, the default settings for direct access for Tomcat and JRun are ports 8080 and 8100 respectively. A URL for JRun is thus http://localhost:8100/webMathematica/Examples/Hello.jsp. The URLs in this document are all specified for Tomcat.

If you do not specify the URL correctly, you may see a "Not Found (404)" error in your browser.

7.1.3 Check the Initial Page

If the initial page that is returned to your browser in response to a web*Mathematica* URL, such as http://localhost:8080/webMathematica/Examples/Hello.jsp, contains some other indication of an error, study it carefully. Some typical problems include configuration errors, such as failure to locate a configuration file or failure to launch *Mathematica*. Studying the initial page and rereading the installation instructions or looking at the specific problems listed later in this section may help to resolve the problem.

If no initial page is returned, your server is not operating. As noted above, if your server does not work, web*Mathematica* cannot work.

©1988-2006 Wolfram Research, Inc. All rights reserved.

If the initial page does not help resolve your problem, please save the page. It may be useful at some later stage.

7.1.4 Check the Kernel Monitor

The kernel monitor contains information on the configuration of the web*Mathematica* site and also can print out information if your server is misconfigured. You should be able to find the monitor via http://localhost:8080/web-Mathematica/KernelMonitor. (You may have some different URL for accessing your server.) The monitor is described in more detail in a previous section.

7.1.5 Check the Log File

The log file is a good place to start to search for information on problems. This is a file to which the servlet container will write log information generated by web*Mathematica*. Under Tomcat, a typical location for the log file is in the `<tomcat-dir>/logs` directory. Under JRun, it is `<jrun-dir>/logs/default-event.log`. For other servlet containers, you will need to study the relevant documentation. An empty log file may indicate that the user running the servlet container does not have permission to write to the log file directory. The Unix command `tail -f` can be useful for monitoring output from this file.

The volume of output is increased if the VerboseLogs property in the `MSP.conf` file is set to `true`. When `VerboseLogs` is `false`, only serious error situations will be reported. Studying the output logs is a good way to uncover problems.

If the log file does not help to resolve your problem, please save it. It may be useful at some later stage.

7.1.6 Check the Console Shell

Some servlet containers are launched from a command-line in a console shell, and this may contain relevant information on any problems.

7.1.7 Check *Mathematica*

Run *Mathematica* in the same way that it is run from the servlet container. For example, under Unix, the servlet container often runs as `tomcat`, so this should be used to run *Mathematica*. Do this for both the *Mathematica* kernel and front end.

Under Unix, a typical command to run the *Mathematica* kernel as `tomcat` is as follows.

```
[user> su -c 'su tomcat -c math'
```

Here is a typical command to run the *Mathematica* front end as `tomcat`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
[user> su -c 'su tomcat -c mathematica'
```

Under Windows, it is possible to run the *Mathematica* kernel and front end from the Start menu.

Running *Mathematica* like this will help to identify problems that may prevent the web system from launching *Mathematica*. One source of problems is caused by a failure to find a license to run *Mathematica*. You can resolve this by making sure that the valid *Mathematica* license is present in the *Mathematica* layout, probably by placing a `mathpass` file into `$TopDirectory/Configuration/Licensing`. The *Getting Started with Mathematica* manual has more information on license problems.

7.2 Specific Problems

This section describes a number of specific problems you might find when installing or running web*Mathematica*. In addition to this section, you may wish to look at the sections on Logging and the Kernel Monitor, as well as the section on Debugging web*Mathematica*.

7.2.1 Problems Running the Front End

Certain operations require the *Mathematica* front end, for example, the rendering of typeset images and graphics or the use of any *Mathematica* notebook API functions. If you are running on a Unix machine and using the X Window System, make sure you have studied the section on Configuring for the X Window System.

7.2.2 Problems Testing Xvnc (Unix only)

This is only an issue for running web*Mathematica* under the X Window System.

As described in the section on installation, it is typical to run a virtual frame buffer X server, Xvnc, to run the *Mathematica* front end. If this does not seem to be running correctly (graphics do not work) you can query the running of the frame buffer by using the `vncviewer` utility.

```
vncviewer :1
```

You will need to enter the password for the Xvnc server and then you will see a view of the screen that the server provides. You should see the *Mathematica* front end running. If there are any problems you may see dialog boxes describing what the problems are.

7.2.3 Problems Testing Xvfb (Unix only)

This is only an issue for running web*Mathematica* under the X Window System.

`Xvfb` is a virtual frame buffer X server that can run the *Mathematica* front end. If it does not seem to be running correctly (graphics do not work) you can query the running of the frame buffer by using the `xwd` utility.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
xwd  -display :1 -root | xwud
```

Under certain configurations, this can be very slow. You can improve the performance by modifying the bit depth of the virtual server, for example, from 24 to 16. The following will run the server with bit depth of 24.

```
su tomcat -c "/usr/X11R6/bin/Xvfb :1 -screen 0 800x600x24 "&
```

The command to run the server with a bit depth of 16 follows.

```
su tomcat -c "/usr/X11R6/bin/Xvfb :1 -screen 0 800x600x16 "&
```

7.2.4 Images Do Not Work

If you find that pages that should hold images, such as the plotting examples, do not actually show any pictures, you should check the log files; problems will be recorded here. Setting `VerboseLogs` to `true` will show the maximum amount of information. It would be good to look at the kernel monitor, since this will tell you the name of the directory into which images are written. If you find that image files are being written in to this directory, but that the URLs that are returned do not locate them, then perhaps you should try to set the web application name as described in a later section.

If you find that typeset images are failing, then you should confirm that the front end is properly configured.

7.2.5 *Mathematica* Packages and Applications

If you find that you have problems using functions from *Mathematica* packages or applications, then study the section on Mathematica Packages and Applications. A problem may occur if you try to use code that does not use the *Mathematica* package format, since the postprocessing code for each request will remove any symbols in the default context.

7.2.6 Kernel Initialization

If you make definitions for symbols that are in the default context with the `KernelInitialization` configuration setting, they will be cleared and the symbols removed by the postprocessing code for each request. This also applies to packages that are loaded from the `KernelInitialization`, which are not written in the *Mathematica* package format to make proper use of contexts. Any definitions must use their own context for names. You can do this by prepending the name with a context (for example `TestNameSpace`Compute)` or by making appropriate use of `BeginPackage[]` and `EndPackage[]`.

Another point about the use of the `KernelInitialization` parameter is that certain packages may require the front end in order to be initialized correctly. You can load these packages into web*Mathematica* with `Developer`UseFrontEnd`; this is shown below.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
KernelInitialization=Developer`UseFrontEnd[Needs[ "MyApplication`"]]
```

7.2.7 Vertical Alignment in Formatting

If you find that formatted output has vertical text (such as superscript, subscripts, or fractions) that does not line up, the problem may be that you are formatting into a text-based output and not using a fixed-width font. The text-based formatting requires a fixed-width font for vertical alignment.

7.2.8 Timeout Problems

You can confirm that your computations are failing to complete, due to the request timing out by inspecting the log system. In this case you should first check the computations. Perhaps there is some problem in the code being executed that causes it to take longer than anticipated. To check this, you could try to run the input in a normal session of *Mathematica*. If you feel the code is running correctly, you could try to increase the `KernelTimeLimit` configuration parameter.

7.2.9 UnsatisfiedLinkError

If you find that web*Mathematica* does not work, you may notice an `UnsatisfiedLinkError` exception in the servlet log.

```
Exception in thread "main" java.lang.UnsatisfiedLinkError:MLOpen
at com.wolfram.jlink.NativeLink.MLOpen(Native Method)
at com.wolfram.jlink.NativeLink.<<init>(Unknown Source)
at
    com.wolfram.jlink.MathLinkFactory.createMathLinkGuts(Unknown=Sourc
    e)
at com.wolfram.jlink.MathLinkFactory.createMathLink(Unknown=Source)
at com.wolfram.jlink.MathLinkFactory.createKernelLinkGuts(Unknown
    Source)
at com.wolfram.jlink.MathLinkFactory.createKernelLink(Unknown=Source)
```

This means that *J/Link* is not installed correctly, specifically that the dynamic library has not been located by the Java system. This library is typically called `libJLinkNativeLibrary.so` on Unix, `JLinkNativeLibrary.dll` on Windows, and `libJLinkNativeLibrary.jnilib` on Mac OS X. Certain servlet containers will not load native libraries from inside a web application. In this case you should copy `SystemFiles` from `WEB-INF/lib` into a general directory and modify the `JLinkNativeLibraryDirectory` configuration parameter as specified here. Many servlet containers, such as Tomcat, can load native libraries from inside a web application. For these, the version of *J/Link* inside web*Mathematica* should work, and if you see this problem you should contact support for assistance.

©1988-2006 Wolfram Research, Inc. All rights reserved.

7.2.10 Cannot Load JLink`

If you find that web*Mathematica* does not work, you may notice in the servlet log that `JLink`` has not been loaded.

```
Error:: Mathematica cannot load JLink`.
    Check that the JLink Mathematica application has been
    installed as described in the JLink documentation.
```

This means that you did not install *J/Link* correctly, specifically that the *Mathematica* application *J/Link* has not been located by *Mathematica*. Since web*Mathematica* contains its own version of *J/Link*, this problem should not be observed, and you should contact support for assistance.

7.2.11 NoClassDefFoundError: TryCatchFinally

If you find that web*Mathematica* does not work, you may notice in the servlet log a report of a `NoClassDefFoundError` exception for `TryCatchFinally`.

```
500 Internal Server Error
/webMathematica/Examples/Hello.jsp:

javax/servlet/jsp/tagext/TryCatchFinally
java.lang.NoClassDefFoundError:
    javax/servlet/jsp/tagext/TryCatchFinally
at java.lang.ClassLoader.defineClass0(Native Method)
at java.lang.ClassLoader.defineClass(ClassLoader.java:509)
at java.lang.ClassLoader.defineClass(ClassLoader.java:438)
```

This error is found when the web*Mathematica* custom JSP tags are used on older servlet containers that do not support the JSP 1.2 API. If you do not wish to upgrade your servlet engine, you can still use web*Mathematica* as described in the section on Classic webMathematica Technology.

7.2.12 NoClassDefFoundError: JLink Classes

If you find that web*Mathematica* does not work, you may notice in the initial web page or in the servlet log a report of a `NoClassDefFoundError` exception. An example is shown below.

```
java.lang.NoClassDefFoundError: com/wolfram/jlink/MathLinkException
          at java.lang.Class.newInstance0(Native Method)
          at java.lang.Class.newInstance(Class.java:237)
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

This means that *J/Link* is not installed correctly; specifically the *J/Link* Java archive has not been located by the Java system. This archive is called `JLink.jar`. Since web*Mathematica* contains its own version of *J/Link*, this problem should not be observed, and you should contact support for assistance.

7.2.13 NoSuchMethodError: KernelData

If you find that web*Mathematica* does not work, you may notice in the initial web page or in the servlet log a report of a `NoSuchMethodError` exception that is generated inside of the `startInit` method of the `KernelData` class. An example is shown below.

```
java.lang.NoSuchMethodError at
    com.wolfram.kerneltools.KernelData.startInit(Unknown Source) at
    com.wolfram.kerneltools.KernelPool.initKernels(Unknown Source) at
    com.wolfram.kerneltools.KernelPoolManager.acquireKernelPool(Unknow
    n Source)
```

This will occur if you try to run web*Mathematica* with an older version of *J/Link*. This can happen, if, at some time in the past, a copy of `JLink.jar` was installed directly into the Java runtime. In general, it is not a good idea to install classes into your Java runtime, because these classes will always be loaded even if a newer version is made available, as happens with this error. The solution is to search in your copy of Java for `JLink.jar` and remove it. You should also search for and remove the native library `JLinkNativeLibrary`, which is often found in the `SystemFiles` directory. web*Mathematica* has its own copy of *J/Link* and there is no need to install *J/Link* into the Java runtime.

7.2.14 Cannot Find jawt.dll (Windows)

On Windows if you find that web*Mathematica* does not work, you may notice a dialog that contains the following information.

```
This application has failed to start because jawt.dll was not found.
```

This happens if the Java runtime cannot find the library `jawt.dll`. The problem can be resolved by amending the PATH environment variable as shown in the section on installing Java. After this you can restart web*Mathematica*.

7.3 Reporting Problems

If you have been unable to resolve your problem, and you are eligible for support, you should prepare the following information. The more detailed the information, the easier it will be to track down the problem.

- The version of web*Mathematica* you are using.
- The version of *Mathematica* you are using.
- Your computer operating system version (e.g., Windows 2000).
- The servlet engine you are using.

©1988-2006 Wolfram Research, Inc. All rights reserved.

- The HTTP server you are using (if applicable, e.g., Apache).
- A one- or two-line summary of the problem, including any steps that may be necessary to reproduce it.

In addition, for installation problems, the following items will be very useful.

- A copy of the initial HTML page.
- A copy of the servlet log file.

This information should then be supplied with any request for support.

If you are not eligible for support, you can report your problem to the web*Mathematica* forum, http://forums.wolfram.com/webmathematica/list/.

©1988-2006 Wolfram Research, Inc. All rights reserved.

8. Classic web*Mathematica* Technology

web*Mathematica* 1.0 provided a different form of HTML templating based on what were known as MSP scripts. While it is recommended that new users use JSPs via the MSP Taglib, MSP scripts are supported in this version of web*Mathematica*. The two technologies are very closely related and it is straightforward to convert from one to the other. This classic version of web*Mathematica* technology may be useful to people who run older servlet containers that do not support the JSP 1.2 API; in this case you will get `NoClassDefFoundError` errors as described above.

8.1 Converting MSP Scripts to JSPs

This section will discuss how to convert between MSP scripts and JSPs that use the MSP taglib. First, most *Mathematica* commands that can be used in an MSP can also be used in a JSP, so it is possible to convert from an MSP to a JSP. The only *Mathematica* command that cannot be used in a JSP is `MSPInclude`. If you need to use this functionality, it should be modified to use `jsp:include`. However, the JSP mechanism is richer than the MSP mechanism, consequently, some JSPs are harder to convert back to MSPs.

8.1.1 Converting an MSP to a JSP

This will be explained with the Plot.msp example, the text of which is shown below.

```
<html>
  <head>
    <title>Plot</title>
  </head>
  <body>
    <h1>Plot</h1>
    <form action="Plot" method="post">
      Enter a function:
      <input type="text" name="fun"  size="24"
        value="<%Mathlet MSPValue[ $$fun, "Sin[x]^2"] %>">
      Enter a number:
      <input type="text" name="x1" size="24"
        value="<%Mathlet MSPValue[ $$x1, "10"] %>">
      <br/>
      <%Mathlet
            MSPBlock[{$$fun,$$x1},
                 MSPShow[Plot[$$fun,{x,0,$$x1}]]] %>
      <hr/>
      <input type="submit" name="btnSubmit" value="Evaluate">
    </form>
  </body>
</html>
```

The corresponding JSP, Plot.jsp, is shown below.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>

<html>
<head>
<title>Plot</title>
</head>
<body bgcolor="#ffffff" >

<form action="Plot.jsp" method="post">

<msp:allocateKernel>
Enter a function:
<input type="text" name="fun"  size="24" value =
    "<msp:evaluate>MSPValue[ $$fun,  "Sin[x]^2"]</msp:evaluate>"
/>
Enter a number:
<input type="text" name="x1"  size="24" value =
    "<msp:evaluate>MSPValue[ $$x1,  "10"]</msp:evaluate>"
/>
<br>
<msp:evaluate>
    MSPBlock[ {$$fun, $$x1},
        MSPShow[ Plot[$$fun, {x,0,$$x1}]]]
</msp:evaluate>
<hr>
</msp:allocateKernel>

<input type="submit" name="btnSubmit" value="Evaluate">

</form>
</body>
</html>
```

The differences are quite apparent and are summarized as follows.

- The script must be renamed to be a JSP.
- The script must be placed in the web*Mathematica* web application (not inside the `WEB-INF` directory).
- The following lines must be added to the top of the script.

```
<%@ page language="java"  %>
<%@ taglib uri="/webMathematica-taglib" prefix="msp" %>
```

- All evaluations must be contained inside a single `<msp:allocateKernel>`.
- `<%Mathlet expr %>` must be replaced with `<msp:evaluate>expr</msp:evaluate>`.
- No change to *Mathematica* computations is needed other than changing uses of `MSPInclude` to use `jsp:include`.
- If *Mathematica* packages are loaded from within the `MSPDirectory`, these should be placed in a directory in `WEB-INF/Applications`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

8.1.2 Converting a JSP to an MSP

It is possible to convert from a JSP that uses the MSP taglib back to an MSP by simply reversing the steps shown in the previous section. This will only work if no other JSP features are being used and if the only MSP taglib tags that are used are `msp:allocateKernel` and `msp:evaluate`.

8.2 Multiple Kernel Pools and MSP Scripts

Multiple kernel pools provide a mechanism for supporting groups of *Mathematica* kernels. Each group has different configuration information and serves different content. These are supported for MSP scripts with the mapping from script name to a pool arranged by setting the name of the pool in the URL. For example, `http://host/webMathematica/MSP/Examples/Script` will find the script `Script.msp` in the pool `Examples`. If no name for a pool is specified, the script will be processed with a *Mathematica* kernel that is taken from the `General` pool. Note that the search for a pool takes place before that for a subdirectory (as described later).

8.3 Processing an MSP Script

This section will describe the different stages that are involved in processing an MSP. As explained previously, MSPs are the original form of web*Mathematica* web technology.

8.3.1 HTTP and MSP Scripts

An MSP script is processed as part of an HTTP transaction. A client sends a request to the server, which replies with a response. One feature of HTTP requests is that they can send parameters and values to the server. This is essential for any dynamic behavior, because parameters are used to select and control the response. The response could be an HTML page. However, it could be some other content type, such as an image, a *Mathematica* notebook, or some form of XML.

8.3.2 The MSP Servlet

A central component of web*Mathematica* technology is the MSP servlet. This is used to process requests and return responses. A URL, such as `http://host/webMathematica/MSP/Script`, will locate the web*Mathematica* web application and then the MSP servlet that it contains. This is enabled by the default configuration file, `web.xml`, which sets up servlet-mapping instructions to direct requests that include MSP in the URL to the MSP servlet.

The MSP servlet deals with each request in a separate thread, allowing more than one request to be processed at the same time. First, it determines the script name, then it gets a *Mathematica* kernel, assigns variables, processes the page or any mathlets in it, performs any postprocessing necessary, and returns the results.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Determine the Script Name

The first task of the MSP servlet is to split up the URL to determine the name of the script. It does this by taking the part of the URL that follows the name of the servlet, which is typically MSP, but which precedes any parameters that have been placed in the URL. This is known as the `PathInfo` and is the equivalent of the CGI variable `PATH_INFO`. A leading or trailing slash (/) character is stripped from the `PathInfo`. If the `PathInfo` contains more than one element, the first element is taken and compared with the names of the kernel pools. If a match is found, then the `PathInfo` (with the name of the pool removed) is used for the MSP script and the corresponding kernel pool is used. If no match is found, then the entire `PathInfo` is used for the name of the MSP script and the `General` pool is used.

Kernel pools are groups of *Mathematica* kernels and are documented in the section on Multiple Kernel Pools. By default, there is only one pool, the `General` pool. The remainder of the URL then indicates the script to be used. This might also contain information on subdirectories. The name of kernel pools or MSPs cannot contain any of the characters such as backslash (\) or colon (:); these are used on certain platforms to indicate path separators. Some examples will now illustrate this process.

For the URL `http://host/webMathematica/MSP/Compute`, the `PathInfo` is `Compute`. This does not contain the name of any pool and hence the script will be `Compute.msp`, and the `General` pool of kernels will be used.

For the URL `http://host/webMathematica/MSP/Examples/Compute`, the `PathInfo` is `Examples/Compute`. If there is a kernel pool named `Examples`, then the script `Compute.msp` from this pool will be used. If there is not a kernel pool called `Examples`, then the script will be `Compute.msp` located in the subdirectory `Examples`.

For the URL `http://host/webMathematica/MSP/Examples/Tests/Compute`, the `PathInfo` is `Examples/Tests/Compute`. If there is a kernel pool named `Examples`, then the script `Compute.msp` from this pool in the subdirectory `Tests` will be used. If there is not a kernel pool called `Examples`, then the script will be `Compute.msp` located in the subdirectory `Examples/Tests`.

It is also possible to specify the name of the script with the variable `$ScriptName`, which you could set on the URL. For example, `http://host/webMathematica/MSP?$ScriptName=Compute` refers to the same script as `http://host/webMathematica/MSP/Compute`.

Get a *Mathematica* Kernel

The MSP servlet then requests a *Mathematica* kernel from the kernel pool that was selected as was described in a previous section. If no pool was named, then the `General` pool will be used. The pool maintains a collection of *Mathematica* kernels waiting for computations. If no kernel is available, the system will wait until one is ready. Using a pool allows the system to share *Mathematica* kernels across multiple requests, which leads to a faster response time for the system.

Note that each request may get a completely different kernel. You cannot rely on saving anything in your *Mathematica* kernel and restoring it the next time.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Assign Input Variables

Any input variables that were sent with the request are then passed to the *Mathematica* kernel with their values. For a variable `sym` and value `fun`, a *Mathematica* assignment `$$sym = "fun"` is made. This makes sure that the value is a *Mathematica* string and cannot evaluate without some special action. Note that `input` elements are not the only sources of variables. For example, an image map can cause variables to be transmitted. web*Mathematica* renames these variables and this helps to ensure that they do not interfere with the *Mathematica* code in your MSPs.

Each variable is scanned to ensure it is a valid *Mathematica* symbol. Any "." character is replaced by a backquote (`), and any underscore (_) is replaced with a "U". These transformations of names are consistent with those in *J/Link*.

Here are some samples of renaming of variables.

Server Variable	*Mathematica Symbol*
`input`	`$$input`
`name.x`	`$$name`x`
`var_x`	`$$varUx`

Each variable is then validated to ensure it only contains letters or digits as well as the dollar ($) and backquote (`) characters. This prevents an attack that sends a variable starting with an exclamation (!) character. This would be potentially dangerous because it might cause *Mathematica* to launch an operating system shell.

Each value is turned into a *Mathematica* string. For this, any backslash (\) and doublequote (") characters are escaped by adding additional backslash (\) characters. If the value starts with an exclamation (!), a space is added. Finally, doublequotes (") are added around the result.

Server Value	*Mathematica Value*
`Sin[x+y]`	`" Sin[x+y] "`
`!myBoolean`	`" !myBoolean"`
`"\foo\bar "`	`" \" \\ foo \\ bar \" "`

Process Page and Mathlets

Interaction with the *Mathematica* kernel starts with a call to `MSPInclude`, which finds the script in the `MSPDirectory` and scans it to find Mathlet tags. The content of each Mathlet is evaluated, which may involve calls to MSP functions that process values and save images. If the result of processing a Mathlet is a string, then this string is returned for inclusion in the HTML page. If it is not a string, it is formatted with `MSPFormat`. This allows a Mathlet to return a string of HTML, but is also convenient since it is not always necessary to return strings.

If any `MSPException` is thrown, it will be caught by the Mathlet processing code, and some suitable error message will be inserted.

If any `MSPReturn` command is evaluated, the processing of the current Mathlet and all other Mathlets is terminated immediately, and the result is returned to the MSP servlet. The content type is saved so that it can be determined by the MSP servlet. If no `MSPReturn` command is encountered, then the contents of the MSP script is

©1988-2006 Wolfram Research, Inc. All rights reserved.

returned with each Mathlet replaced by the result of its evaluation. In the absence of MSPReturn, the default content type is text/html, though this can be changed by a setting of the ContentType option of MSPPageOptions.

The processing of each Mathlet is wrapped in TimeConstrained, the time parameter being computed from the KernelTimeLimit configuration parameter.

If *Mathematica* generates any messages while processing the page, they will be sent across the *MathLink* connection back to the MSP servlet. This uses a *MathLink* packet listener to catch any messages and send them to the MSP logging system. If this is running in a verbose mode, the messages will be displayed. The documentation for *J/Link* describes packet listeners.

Postprocess

After the result is returned by the kernel, the MSP servlet calls the kernel to determine the content type of the result. Typically this will be text/html, but it could be something different—for example, application/mathematica for a *Mathematica* notebook or image/gif for a GIF image. These different content types can be generated by the MSP function MSPReturn or by a setting of the ContentType option of MSPPageOptions.

$ContextPath is now reset to its value before the page was loaded. Using Needs can prevent loading a package for every page request. *The Mathematica Book* contains more details on the operation of Needs.

The kernel is then cleaned by clearing the values of parameters that were sent with the request and by removing all symbols in the default context. In addition, any Java object references are removed, and any streams that were opened are closed. The kernel is then returned to the kernel pool and can be used by the next request.

Return the Response

A response to the client is made with the result and content type. The request is now complete.

Process Images

One modification to this description concerns images. If an MSPFormat or MSPShow generates an image, it will be stored in a file and an img tag that references the file returned.

```
<img
     src="/webMathematica/MSP?MSPStoreID=MSPStore1043815006_0&amp;MSPSt
     oreType=image/gif" alt="Created by webMathematica" />
```

The SRC attribute references the MSP servlet through a relative URL and passes a parameter that gives the name of the file. The MSP servlet returns the contents of the file and makes periodic passes that delete old image files.

The configuration file contains the name of the directory in which images are saved by the setting ImageDirectory. If you find that plots or other images are not working, you should look in the log files (described in the section on Logging) to see if there are problems associated with the image directory.

©1988-2006 Wolfram Research, Inc. All rights reserved.

8.3.3 Overview

An overview of the steps in processing a page can be found in the introductory section on web*Mathematica* Technology.

8.4 Mapping URLs onto MSP Scripts

For users of web*Mathematica* who write their content in MSP scripts, they are located inside the web*Mathematica* web application, in the directory `/webMathematica/WEB-INF/MSPScripts`. This can be modified in the configuration file `MSP.conf` with the setting `MSPDirectory`. Of course, you may not want to keep all MSP scripts in the same directory, preferring to place them in a variety of different locations. This section discusses some of the ways that this can be done.

8.4.1 Script Directories

A URL, such as `http://host/webMathematica/MSP/Script`, will find the MSP `Script.msp` in the main directory. You can also place MSP scripts in subdirectories of the main directory. For example, a URL, such as `http://host/webMathematica/MSP/Test/Script1`, will locate `Script1.msp` in the subdirectory `Test`. When an MSP is loaded from a subdirectory, that subdirectory is added to the *Mathematica* `$Path` so that *Mathematica* packages and applications located in the subdirectory can be loaded with no need to specify the name of the subdirectory. Many of the MSP scripts located in the `Examples` directory that ship with web*Mathematica* make use of this feature.

The following two tables give a summary on loading MSP scripts from the main directory and from subdirectories.

In the first table, the MSP `Script.msp` is located in the main directory, `/usr/local/tomcat/webapps/webMathematica/WEB-INF/MSPScripts`. It can be accessed by the URL `http://host/webMathematica/MSP/Script`. The configuration parameter `MSPDirectory` specifies the location of this directory, which is added to the *Mathematica* `$Path`.

	Main Directory
Script Name	`Script.msp`
Script Location	`/usr/local/tomcat/webapps/webMathematica/WEB-INF/MSPScripts`
URL	`http://host/webMathematica/MSP/Script`

In this second table, `Script1.msp` is located in a subdirectory of the main directory, `/usr/local/tomcat/webapps/webMathematica/WEB-INF/MSPScripts/Test`. The URL `http://host/webMathematica/MSP/Test/Script1` will find this MSP script. In the *Mathematica* kernel, the directories `/usr/local/tomcat/webapps/webMathematica/WEB-INF/MSPScripts` and `/usr/local/tomcat/webapps/webMathematica/WEB-INF/MSPScripts/Test` are added to `$Path`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

	Subdirectory
Script Name	`Script1.msp`
Script Location	`/usr/local/tomcat/webapps/webMathematica/WEB-INF/MSPScripts/Test`
URL	`http://host/webMathematica/MSP/Test/Script1`

Note that the same kernel pool serves the MSPs in the main and subdirectories. Consequently, they all have the same configuration details.

8.5 MSP Scripts and Static Files

MSPs process static files in a special way because the relative URL actually refers to the MSP servlet, which reads and returns the appropriate content. The file extensions that the MSP servlet recognizes as valid static content are set by the configuration parameter StaticFileExtensions. The default value is as follows.

```
StaticFileExtensions=gif:image/gif,jpg:image/jpeg,jpeg:image/jpeg
```

If you do not wish to serve any static files from your site, you can omit this definition. Alternatively, you could add extensions for other formats, such as sound or scripting archives.

8.6 Example MSP Scripts

These are a collection of sample MSP scripts, each of which will demonstrate some detail or feature. The sources for all these examples are included in the *Mathematica* MSP application in the directory `MSPScripts/Examples`.

8.6.1 A Beginning: Hello.msp

If you installed web*Mathematica* as described above, you should be able to connect to this MSP script via http://localhost:8080/webMathematica/MSP/Examples/Hello. (You may have some other URL for accessing your server.)

This example evaluates the `Date[]` function of *Mathematica*. The result changes each time the page is accessed, demonstrating that this really is a dynamic process. The server sees the request for `Hello` and searches for an MSP called `Hello.msp`. The source for this page is in `MSPScripts/Examples/Hello.msp`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<html>
  <head>
    <title>Hello World (MSP)</title>
  </head>
  <body>
    <h1>Hello World (MSP)</h1>
    <h4>Date[]</h4>
    <%Mathlet Date::usage %>
    <p>Its current value is:</p>
    <%Mathlet Date[] %>
  </body>
</html>
```

This is standard HTML, except for the two Mathlet tags `<%Mathlet expr %>`. *Mathematica* replaces each Mathlet with the result of evaluating and formatting its contents. This builds a new HTML document, which is then returned to the client.

Of course, this example is not interesting, because the user is not doing anything other than retrieving a page. For interactive behavior, a user needs to send input values with the request. The next example, Variables.msp, discusses this further.

8.6.2 Working with Variables: Variables.msp

If you installed web*Mathematica* as described earlier, you should be able to connect to this MSP script via http://localhost:8080/webMathematica/MSP/Examples/Variables. (You may have some other URL for accessing your server.) It demonstrates how variables are connected to input values. The source for this page is in `MSPScripts/Examples/Variables.msp`.

```
<html>
  <head>
    <title>Assigning Variables</title>
  </head>
  <body>
    <h1>Assigning Variables</h1>
    <form action="Variables" method="post">
      Enter something:
      <input type="text" name="input"  size="10">
      <br>
      <%Mathlet $$input %>
      <br>
      <input type="submit" name="submitButton" value="Evaluate">
    </form>
  </body>
</html>
```

This page is more elaborate because it contains `form` and `input` elements. These are important ways for allowing interaction from the client.

©1988-2006 Wolfram Research, Inc. All rights reserved.

A form element is a block of HTML that may contain input elements. A form may be activated with an input of type submit; this sends the name and value associated with each input tag to the server. Here, the opening tag of the form element contains two attributes. The action attribute refers to a URL that is accessed when the form is activated. In this case, it is a relative URL that refers to the original Variables script. The method attribute tells the browser what HTTP method to use—in this case, a post method. (It is very common to use post methods.)

This example has two input tags: the first allows the user of the page to enter text, and the second specifies a button that, when pressed, will submit the form. When the form is submitted, it will send information from input elements to the URL specified by the action attribute (in this case, the same MSP). Information entered by the user is sent to a *Mathematica* kernel and assigned to a *Mathematica* symbol. The name of the symbol is given by prepending $$ to the value of the name attribute. Here the name attribute is input, and the *Mathematica* symbol $$input is assigned to the contents of the input element.

The first time the page is accessed there is no value for $$input. When a value is entered in the text field and the Evaluate button pressed, the text is displayed. Note that the value is a *Mathematica* string—if you enter an input such as "5+7", no computation is actually done. If you want the input to be interpreted and evaluated by *Mathematica*, you need to use one of the MSP functions.

You can distinguish between *Mathematica* symbols that get definitions from the server versus other symbols because the former all receive a $$ prefix. This is necessary to make sure that symbols from the server, such as those from an image map, do not collide with regular *Mathematica* symbols.

8.6.3 Using MSP Functions: Expand.msp

If you installed web*Mathematica* as described above, you should be able to connect to this MSP script via http://localhost:8080/webMathematica/MSP/Examples/Expand . (You may have some other URL for accessing your server.) It allows a polynomial and an exponent to be entered. When the submit button is pressed, the polynomial is raised to the power and expanded, and an HTML page that contains the result is returned. The source for this page is in MSPScripts/Examples/Expand.msp.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<html>
  <head>
    <title>Expanding Polynomials</title>
  </head>
  <body>
    <h1>Expanding Polynomials</h1>
    <form action="Expand" method="post">
      Enter a polynomial (e.g. x+y):
      <input type="text" name="expr"  size="10">
      Enter a positive integer (e.g. 4):
      <input type="text" name="num" size="3">
      <br>
      <%Mathlet
             MSPBlock[{$$expr,$$num},
                      Expand[$$expr^$$num]] %>
      <br>
      <input type="submit" name="submitButton" value="Evaluate">
    </form>
  </body>
</html>
```

This page contains `form` and `input` tags as described in the previous example. Additionally, the Mathlets refer to *Mathematica* functions that are not in standard usage.

When the `form` is submitted, the server connects to a *Mathematica* kernel, in which two symbols, `$$expr` and `$$num`, are assigned to the text from the two `input` elements. If no text is entered, the symbols will not have any definition.

Mathematica now evaluates the Mathlet. The `MSPBlock` command is a programming construct, which here inspects two variables, `$$expr` and `$$num`. If either of these has no value, `MSPBlock` returns a null string, which is why the first time you access the page, you do not see a result. The values of both variables are then interpreted by *Mathematica*. If successful, the results of interpretation are substituted into the second argument or body of `MSPBlock`. In this example, all instances of `$$expr` are substituted with the parsed value of `$$expr`, and the same is done for `$$num`. The result is then evaluated, formatted, and placed in the HTML page, which is returned to the client.

Interpretation of the variables by *Mathematica* can fail in two ways: the input might not be valid *Mathematica* input (for example, `f[`), or it might be some dangerous input (such as `ReadList["/etc/passwd"]`). In both cases, the inputs are rejected and an error message generated. This demonstrates some of the security features of the system, which the Security section documents in detail.

8.6.4 Graphics: Plot.msp

If you installed web*Mathematica* as described above, you should be able to connect to this MSP script via http://localhost:8080/webMathematica/MSP/Examples/Plot. (You may have some other URL for accessing your server.) It allows the user to enter a function to be plotted. The source for this page is in `MSPScripts/Examples/Plot.msp`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<html>
  <head>
    <title>Plot</title>
  </head>
  <body>
    <h1>Plot</h1>
    <form action="Plot" method="post">
      Enter a function:
      <input type="text" name="fun" size="24"
        value="<%Mathlet MSPValue[ $$fun, "Sin[x]^2"] %>">
      Enter a number:
      <input type="text" name="x1"  size="24"
        value="<%Mathlet MSPValue[ $$x1, "10"] %>">
      <br/>
      <%Mathlet
            MSPBlock[{$$fun,$$x1},
                 MSPShow[Plot[$$fun,{x,0,$$x1}]]] %>
      <hr/>
      <input type="submit" name="btnSubmit" value="Evaluate">
    </form>
  </body>
</html>
```

In this example, the two text `input` elements use Mathlets to fill in values when the page is first requested. It is often useful to suggest input and also to preserve input that was previously entered. These use `MSPValue`, which returns the value of the variable in the first argument, or the second argument if the variable has no value. This demonstrates a new use of a Mathlet: to provide inline expansion for a quoted attribute value. Mathlets are evaluated completely out of context for the syntax of the page, and can go anywhere, supplying any part of the text of the page.

The third Mathlet then evaluates. The `MSPBlock` makes sure that both `$$fun` and `$$x1` have valid values, which are used in the `Plot` command. `MSPShow` saves an image on the server and returns the necessary HTML to refer to this image. You may wish to use the View Source option of your browser to see how the page refers to the image. The image uses a GIF format; it is possible to save images in other formats, such as JPEG, by setting a third argument. The MSP Function Reference section documents this feature.

8.6.5 Typeset Images: Integrate.msp

If you installed web*Mathematica* as described above, you should be able to connect to this MSP script via http://localhost:8080/webMathematica/MSP/Examples/Integrate. (You may have some other URL for accessing your server.) It allows the user to enter a function to be integrated. The result is then formatted by the typesetting system and saved as an image. The source for this page is in `MSPScripts/Examples/Integrate.msp`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<%Mathlet
    integrand = Null;
    If[ MSPValueQ[ $$expr],
        integrand = MSPToExpression[ $$expr]] ;
%>

<html>
  <head>
    <title>Integration</title>
  </head>
  <body>
    <h1>Integrate a function</h1>
    <form action="Integrate" method="post">Input:
      <br>
      <input type="text" name="expr" size="24"
          value="<%Mathlet MSPValue[ $$expr, "Sin[x]^2"] %>">
      <br/>
      <%Mathlet
        If[ integrand =!=
     Null,MSPFormat[Integrate[integrand,x],StandardForm] %>
      <br/>
      <input type="submit" name="btnSubmit" value="Evaluate">
    </form>
  </body>
</html>
```

In this example, a Mathlet integrates an expression and uses `MSPFormat` to format the result with `StandardForm`. This generates an image and returns a reference to the image. For this to work, it is necessary to use the *Mathematica* front end.

The example also demonstrates the use of page scoped variables with `MSPToExpression`. This is an alternative to the use of `MSPBlock` that is a bit more complicated but also more expressive. The page variable `integrand` is initialized to `Null`, and later, if its value has been modified, the integration is carried out. It is assigned to the interpreted value of `$$expr` only if this input variable actually has a value. Note that if an error, such as a security error, is encountered in interpreting `$$expr`, an exception will be thrown and `integrand` will not be assigned to a new value. An alternative to scoping a variable to a page is to scope to a session, which is possible with `MSPSessionVariable`.

8.6.6 Live 3D Plotting: Plot3DLive.msp

If you installed web*Mathematica* as described above, you should be able to connect to this MSP script via http://localhost:8080/webMathematica/MSP/Examples/Plot3DLive . (You may have some other URL for accessing your server.) It allows the user to enter a function to be plotted by the `LiveGraphics3D` applet. The source for this page is in `MSPScripts/Examples/Plot3DLive.msp`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<html>
  <head>
    <title>Live 3D Plotting</title>
  </head>
  <%Mathlet $ImageBackground = "#ffffff"; $ImageSize = {300,300}; %>
  <body  bgcolor = "#ffffff">
    <h1>Live 3D Plotting</h1>
    <form action="Plot3DLive" method="post">
      Plot3D of
      <input type="text" name="fun" size="22" value =
        "<%Mathlet MSPValue[ $$fun, "Sin[x y]^2"] %>">
      <br/>
      x from:
      <input type="text" name="x0"  size="10" value =
         "<%Mathlet MSPValue[ $$x0, "-2"] %>">
      to:
      <input type="text" name="x1"  size="10" value =
         "<%Mathlet MSPValue[ $$x1, "2"] %>">
      <br/>
      y from:
      <input type="text" name="y0"  size="10" value =
          "<%Mathlet MSPValue[ $$y0, "-2"] %>">
      to:
      <input type="text" name="y1" size="10" value =
          "<%Mathlet MSPValue[ $$y1, "2"] %>">
      <br/>
      Number of points to plot:
      <input type="text" name="pts"  size="5" value =
          "<%Mathlet MSPValue[ $$pts, "20"] %>">
     <br/>
     <%Mathlet
         MSPBlock[ {$$fun, $$x0, $$x1, $$y0, $$y1, $$pts},
            MSPLive3D[
                Plot3D[$$fun, {x,$$x0,$$x1}, {y,$$y0,$$y1},
                     PlotPoints -> $$pts]]] %>
     <br/>
     <input type="submit" name="btnSubmit" value="Evaluate">
    </form>
  </body>
</html>
```

This example uses a number of Mathlets to set up parameters. The last Mathlet takes the values of these parameters and uses them in a call to `Plot3D`. The result of this is used by `MSPLive3D` to call the `LiveGraphics3D` applet. This gives a real-time rotation of the three-dimensional graphics object.

8.6.7 Returning General Content: Content.msp

If you installed web*Mathematica* as described above, you should be able to connect to this MSP script via http://localhost:8080/webMathematica/MSP/Examples/Content. (You may have some other URL for accessing your server.)

©1988-2006 Wolfram Research, Inc. All rights reserved.

All of the examples up to this point return HTML to the browser, but the web can work with general content involving many different formats. MSPReturn is provided to allow an MSP script to return arbitrary content. Here is an example that demonstrates how different formats can be returned. The source is in MSPScripts/Examples/Content.msp and MSPScripts/Examples/Content.m.

First, here is the source of the MSP script.

```
<html>
  <head>
    <title>General Content</title>
  </head>
  <body bgcolor="#ffffff">
    <h1>General Content</h1>
      <form action="Content" method="post">
        <p>
          This example generates a notebook, converts it into a
   specific format and returns
          that format.
        </p>
        <%Mathlet
          If[ MSPValueQ[ $$button],
              Get[ "Content.m"];
              MSPReturn @@ GeneralContent[ $$button]] %>
        <br>
       <input type="Submit" name="button" value="Notebook">
       <input type="Submit" name="button" value="PostScript">
       <input type="Submit" name="button" value="GIF">
     </form>
  </body>
</html>
```

Here is the *Mathematica* source.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
MakeNotebook[] :=
    Developer`UseFrontEnd[
        Module[ {nb, nbobj},
            nb = NotebookCreate[] ;
            NotebookWrite[ nb, Cell[ "A Dynamically Created
    Notebook", "Title"]] ;
            NotebookWrite[ nb,
                Cell[ "Converted to " <> $$button, "Subtitle"]] ;
            NotebookWrite[ nb, Cell[ "The date is " <> ToString[
    Date[]], "Text"]] ;
            nbobj = NotebookGet[ nb] ;
            NotebookClose[ nb] ;
            nbobj]]


GeneralContent[ fmt_] :=
    Module[ {nbobj},
        nbobj = MakeNotebook[] ;
        Developer`UseFrontEnd[
            Switch[
                fmt,
                    "Notebook",
                        {ToString[ nbobj, InputForm],
                                "application/mathematica"},
                    "PostScript",
                        {DisplayString[ nbobj, "EPS"],
                                "application/eps"},
                    "GIF",
                        {DisplayString[ nbobj, "GIF"],
                                "image/gif"},
                    _, "Unknown format"]
                ]]
```

In this example one evaluation tests the variable `$$button`. If it has a value from activating one of the buttons in the `form`, this is used to specify a return format type and passed to a function `GeneralContent`. The *Mathematica* code for this function is placed into a separate package to be loaded when the variable is set. `GeneralContent` calls a function that creates a simple notebook, `MakeNotebook`. `MakeNotebook` generates a notebook using the *Mathematica* notebook API and the function `Developer`UseFrontEnd`. In a real life situation, it would be written to build a more interesting notebook. `MSPReturn` returns the representation of the notebook to the server with the content type. This is then returned to the browser, which, if suitably configured, will deploy the necessary helper application.

In a real example, a dynamically generated notebook would probably use information sent with the request from the client.

If you wish to return special content and also set a filename to be used with this content, then you may wish to use the three-argument form of `MSPReturn`. This is described in the *Mathematica* Function Reference section.

Another way to set the content returned from an MSP script is to use `MSPPageOptions`. The topic of returning general content was discussed earlier.

©1988-2006 Wolfram Research, Inc. All rights reserved.

8.6.8 Applets: TextApplet.msp

This example demonstrates how to call on the services of a *Mathematica*-powered web site from an applet. This shows a combination of client and server programming. The section involves some programming in Java.

If you installed web*Mathematica* as described previously, you should be able to connect to this MSP script via http://localhost:8080/webMathematica/MSP/Examples/TextApplet. (You may have some other URL for accessing your server.) The source for this page is in `MSPScripts/Examples/TextApplet.msp` and in `webMathematica/WEB-INF/src/ExampleApplets/TextApplet.java`.

First, here is the source of the MSP script.

```
<%Mathlet
    If[ MSPValueQ[ $$Compute],
      MSPReturn[ "Date[] returns " <> ToString[ Date[]],
     "text/plain"]] %>
<html>
  <title>Applet Test</title>
  <body>
    <p>
      Here is an applet that gets a result from Mathematica:
    <br>
    <applet CODE="TextApplet.class"
        archive = "<%Mathlet $WebApplication <>
     "/Resources/applets/MSPExamples.jar" %> "
        width=400
        height=30>
      <param name="ArgumentURL" Value="TextApplet?Compute=True">
    </applet>
    <p>
      Hitting refresh will cause the page to update.
  </body>
</html>
```

Here is the source for the applet, `TextApplet.java`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
import java.applet.Applet;
import java.awt.*;
import java.net.*;
import java.io.*;
public class TextApplet extends Applet
{
  public void paint(Graphics g)
  {
    super.paint(g);
    try {
      URL url=new URL(getDocumentBase(),getParameter("ArgumentURL"));
      InputStream in=url.openStream();
      ByteArrayOutputStream out=new ByteArrayOutputStream();
      byte[] b=new byte[1024]; int len;
      while((len=in.read(b,0,1024)) != -1) {
        out.write(b,0,len);
      }
      b=out.toByteArray();
      g.drawBytes(b,0,b.length-1,20,20);
    }
    catch (Exception e) {
      System.out.println("Error "+e);
    }
  }
}
```

This is a very simple applet; the paint method opens a connection to a URL, the name of which is formed from the document that loaded the applet, and the value of the parameter ArgumentURL, which is passed in from a `param` tag. This causes the TextApplet MSP script to be called and return a computation of the date.

8.6.9 Calling Back to Java: Request.msp

This example demonstrates how an MSP script can call on Java, a feature provided by *J/Link*. This allows *Mathematica* to call Java and Java to call *Mathematica*. It also allows *Mathematica* called from Java itself to call back into Java. The *J/Link* manual contains more information on this feature. This example requires some knowledge of Java.

The example demonstrates how to call methods on the `HttpServletRequest` object, a Java object used by the servlet engine to hold information on the HTTP request. You can learn more about the `HttpServletRequest` from books on servlets or by looking at the servlet API, documented at http://java.sun.com/products/servlet/.

If you installed web*Mathematica* as described above, you should be able to connect to this MSP script via http://localhost:8080/webMathematica/MSP/Examples/PlotScript. (You may have some other URL for accessing your server.) The source for this page is in `MSPScripts/Examples/PlotScript.msp` and in `MSPScripts/Examples/PlotScript1.msp`.

Here is the source for `Request.msp`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<html>
<head>
<title>Request Information</title>
</head>
<body bgcolor="#ffffff">

<h1>Request Information</h1>

<h4>HTTP Method</h4>

<%Mathlet $ServletRequest@getMethod[] %>

<h4>Header Names and Values</h4>

<%Mathlet
    JavaBlock[
        Block[
            {headers, res, $PathWidth = 100},
            headers = $ServletRequest@getHeaderNames[] ;
            res = Rest[
                NestWhileList[
                    Module[ { name = headers@nextElement[]},
                        {name, $ServletRequest@getHeader[name]}] &,
                    1, headers@hasMoreElements[] &]] ;
            MSPFormat[ TableForm[res ],OutputForm]
            ]
        ]
     %>
</body>
</html>
```

The first Mathlet calls the `getMethod` method of the `$ServletRequest` object, and this returns the HTTP method, such as GET, that was used. The second Mathlet has some more sophisticated code that uses a Java `Enumeration` (returned by `getHeaderNames`) to extract the names and values of the headers. This is returned, formatted by `TableForm`.

8.6.10 Uploading a File: UploadFile.msp

This example demonstrates how to upload a file to a web*Mathematica* server.

If you installed web*Mathematica* as described earlier, you should be able to connect to this MSP script via http://localhost:8080/webMathematica/MSP/Examples/UploadFile . (You may have some other URL for accessing your server.) There are two scripts involved. The first has a form for submitting the file, and the second reads the file and displays its contents. The sources are in `MSPScripts/Examples/UploadFile.msp` and `MSPScripts/Examples/UploadFileAction.msp`. A portion of `UploadFile.msp` is shown below.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<form method=post enctype="multipart/form-data"
    action="UploadFileAction">
<p>
Enter a file to upload:
</p>
<br/>
    <input type="file" size=40 name="file">
<br/>
<br/>
    <input type="reset" value="Clear">
    <input type="submit" value="Submit">
</form>
```

This is really an HTML page since it contains no *Mathematica* computations. It is written as an MSP so that it can reference `UploadFileAction.msp` conveniently. It uses a `multipart/form-data` form element, which is the standard way to submit a file. The form is submitted to `UploadFileAction.msp`, which is shown below.

```
<html>
<head>
<title>Upload File Action</title>
</head>
<body>
<p>
File contents:
</p>
<br>
<pre>
<%Mathlet
    file = "FileName" /. MSPGetUploadFile[];
    Read[ file, Record, RecordSeparators-> {}]
%>
</pre>

<p>
<a href="UploadFile">Upload another file?</a>
</p>

</body>
</html>
```

This uses the function `MSPGetUploadFile`, which uploads the file and saves it on the server. It returns a list of useful information, including the filename which is used on the server, the original filename used on the client, and the content-type. In a case where there is more than one file to be uploaded, `MSPGetUploadFile` will throw an exception. If you wish to upload more than one file, you can use `MSPGetUploadFileList`. In this example, the contents of the file are then read and displayed. Of course, it would be simple to use some other importing function and then do some computation.

©1988-2006 Wolfram Research, Inc. All rights reserved.

9. Appendix

9.1 *Mathematica* Function Reference

A number of *Mathematica* commands are available for inserting into JSPs that use tags from the MSP taglib. This section gives a description of each command. It is possible to use and test these commands by using an interactive *Mathematica* session. Of course, they are designed to work while a web page is being processed, but it can still be useful to get some idea of how they work in a *Mathematica* session.

9.1.1 MSPBlock

MSPBlock[{var_1, ...}, *body*]	interpret the argument variables and replace in the body
MSPBlock[{var_1, ...}, *body*, *defvalue*]	if any of the variables do not have values, *defvalue* is returned

This is one of the key ways to work with variables from the HTTP request. MSPBlock takes each of the variables var_i, interprets them, and then replaces any occurrences in *body* with the interpreted value. If any variables do not have values, an empty string is returned. If a security or parse error is encountered when interpreting the value, an MSPException is thrown.

You can simulate how the functions work by loading the package and setting the security content.

```
In[1]:= Needs["MSP`"];

In[2]:= SetSecurity[];
```

Here the variable $$var is assigned to the value "5+7"; note that the value is a string.

```
In[3]:= $$var = "5+7";
```

When MSPBlock evaluates, all occurrences of $$var are replaced by its interpreted value.

```
In[4]:= MSPBlock[ {$$var}, {Hold[ $$var], $$var}]

Out[4]= {Hold[5 + 7], 12}
```

If the input value cannot be interpreted, an MSPException is thrown.

```
In[5]:= $$var = "Sin[";

In[6]:= Catch[ MSPBlock[ {$$var}, {Hold[ $$var], $$var}], _MSPException, List]
        ToExpression::sntx : Syntax error in or before "Sin[".

Out[6]= {{$$var, Sin[}, MSPException[ParseError]}
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

If the input value does not pass the security test, an MSPException is thrown.

```
In[7]:= $$var = "ReadList[\"/etc/passwd\"]";

In[8]:= Catch[ MSPBlock[ {$$var}, $$var], _MSPException, List]

Out[8]= {{$$var, ReadList["/etc/passwd"]}, MSPException[SecurityError]}
```

Input can also be given in MathML.

```
In[9]:= $$e = "<math><mrow><mi>sin</mi><mo>&af;</
            mo><mrow><mo>(</mo><mi>x</mi><mo>)</mo></mrow></mrow>
            </math>";

In[10]:= MSPBlock[ {$$e}, $$e]

Out[10]= Sin[x]
```

Exceptions

MSPBlock can throw the following exceptions.

MSPException["ParseError"] if the value cannot be interpreted by *Mathematica*.

MSPException["SecurityError"] if the value does not pass the security test.

MSPException["ValueError"] if the value is not a string. This would indicate a programmatic error by the page author.

MSPException["VariableError"] if the variable is not a *Mathematica* symbol. This would indicate a programmatic error by the page author.

9.1.2 MSPException

MSPException[ParseError]	thrown if a value cannot be parsed by *Mathematica*
MSPException[SecurityError]	thrown if the parsed result does not pass the security check
MSPException[NoValueError]	thrown by MSPToExpression if a variable has no value
MSPException[VariableError]	thrown if an illegal variable (a variable that is not a *Mathematica* symbol) is encountered
MSPException[ParseError]	thrown if an illegal value (a value that is not a *Mathematica* string) is encountered
MSPException[VersionError]	thrown if MSPPageOptions detects an attempt to process a page that needs a higher version of the MSP technology

A number of MSP commands throw an MSPException when some error situation occurs. These are caught by the page processing code, but it would be permissible for a page author to catch them and process them in some intermediate step.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
In[1]:= Needs["MSP`"];
```

If a variable cannot be interpreted, a ParseError exception is thrown. Since the values may be entered from the client, this does not indicate an author error.

```
In[2]:= Catch[ MSPToExpression["f["], _MSPException, List]

        ToExpression::sntx : Syntax error in or before "f[".

Out[2]= {{f[, f[}, MSPException[ParseError]}
```

If the result of interpretation does not pass the security check, a SecurityError exception is thrown. Since the values may be entered from the client, this does not indicate an author error.

```
In[3]:= Catch[ MSPToExpression["ReadList[\"/etc/passwd\"]"], _MSPException, List]

Out[3]= {{ReadList["/etc/passwd"], ReadList["/etc/passwd"]},
          MSPException[SecurityError]}
```

If a variable that is not a *Mathematica* symbol is encountered, a VariableError exception is thrown. This usually indicates an author error.

```
In[4]:= Catch[ MSPValue[x[1]], _MSPException, List]

Out[4]= {x[1], MSPException[VariableError]}
```

If a value that is not a *Mathematica* string is encountered, a ValueError exception is thrown. This usually indicates an author error.

```
In[5]:= $$var = Sin[1]

Out[5]= Sin[1]

In[6]:= Catch[ MSPToExpression[$$var], _MSPException, List]

Out[6]= {{$$var, Sin[1]}, MSPException[ValueError]}
```

9.1.3 MSPExportImage

MSPExportImage[*expr*]	use the *Mathematica* Export command to save an image in GIF format and return an img tag
MSPExportImage[*expr*, *format*]	use the *Mathematica* Export command to save an image in a specified format and return an img tag

MSPExportImage is used to save an image using the *Mathematica* command Export. If the first argument is a graphics object or a Cell or BoxData object, it is displayed directly. If it is a general expression, it is formatted with TraditionalForm. A further discussion on formatting mathematics and graphics is given in the section on Displaying Mathematics and Graphics.

```
In[1]:= Needs["MSP`"];

In[2]:= MSPExportImage[ Graphics[ Line[ {{0, 0}, {1, 1}}]]]
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
Out[2]=  <img src="/webMathematica/MSP?MSPStoreID=FileNameBase_245528838&
          MSPStoreType=image/gif" alt="Created by webMathematica" />
```

Here MSPExportImage receives a box expression to be rendered into an image.

```
In[3]:= MSPExportImage[ BoxData[ToBoxes[ x + y^2, TraditionalForm]]]

Out[3]=  <img src="/webMathematica/MSP?MSPStoreID=FileNameBase_574601872&
          MSPStoreType=image/gif" alt="Created by webMathematica" />
```

In this example MSPExportImage formats its argument with TraditionalForm before rendering it into an image.

```
In[4]:= MSPExportImage[ x + Sin[x]]

Out[4]=  <img src="/webMathematica/MSP?MSPStoreID=FileNameBase_932131661&
          MSPStoreType=image/gif" alt="Created by webMathematica" />
```

Here MSPExportImage uses its second argument to specify the image format type.

```
In[5]:= MSPExportImage[ x + Sin[x], "JPEG"]

Out[5]=  <img src="/webMathematica/MSP?MSPStoreID=FileNameBase_665363422&
          MSPStoreType=image/jpeg" alt="Created by webMathematica" />
```

Since this uses Export, it is possible to set conversion options to control how the image is generated. The actual option settings are documented in the *Mathematica* Help Browser for Export under Further Information. For example, this generates a GIF image for which any white colors will be transparent.

```
In[6]:= MSPExportImage[ BoxData[ToBoxes[ x + y^2]], "Transparency" → GrayLevel[1]]

Out[6]=  <img src="/webMathematica/MSP?MSPStoreID=FileNameBase_787204206&
          MSPStoreType=image/gif" alt="Created by webMathematica" />
```

Default values for these options can be set with the variable $ExportImageOptions. For example, this causes all images to use white as the transparent color.

```
In[7]:= $ExportImageOptions = "Transparency" → GrayLevel[1]

Out[7]= Transparency → GrayLevel[1]
```

9.1.4 MSPFormat

MSPFormat[*expr*]	format *expr* in the format style $MSPFormatType
MSPFormat[*expr*, *fmt*]	format *expr* in the format style *fmt*
MSPFormat[*expr*, *fmt*, *type*]	format *expr* in the format style *fmt*, using *type* as the content type

This is one of the important ways to format results from *Mathematica*. A more general discussion of output is available in the section on Evaluation Formatting and a discussion on formatting mathematics and graphics is given in the section on Displaying Mathematics and Graphics.

©1988-2006 Wolfram Research, Inc. All rights reserved.

The formatted result can appear in the different format types that *Mathematica* provides for output. These include OutputForm, InputForm, StandardForm, TraditionalForm, and MathMLForm. In addition the result can be returned as HTML, an image format, or as MathML. You can learn more about *Mathematica* formatting in *The Mathematica Book*. The second argument of MSPFormat is a symbol that selects the *Mathematica* format type, and the third argument is a string that sets the actual content type of the result.

The result of any Mathlet that is not a string is formatted with MSPFormat. You can prepare your own output by generating your own string and not invoking MSPFormat.

In order for the following examples to work, the MSP application must be loaded.

```
In[1]:= Needs["MSP`"]
```

HTML Formatting

MSPFormat will return content in HTML for OutputForm and InputForm. These are string-based formatting types. The result is correctly escaped to be valid HTML that will work in a web page. Note that the result must be displayed in a fixed-width font for correct alignment of multiline output.

```
In[10]:= MSPFormat[ x + y^2, OutputForm]

Out[10]= <p> <pre>
         &amp;#160;&amp;#160;&amp;#160;&amp;#160;&amp;#160;2
         x&amp;#160;+&amp;#160;y
         </pre></p>
```

Expressions are formatted to a page width set by the variable $PageWidth. An alternative way to format expressions into HTML is provided by the HTML Functions.

Image Formatting

MSPFormat uses an image content for StandardForm and TraditionalForm. It creates and saves an image in a file on the server. An img tag, which can be used to retrieve the image, is then returned as the result.

```
In[3]:= MSPFormat[ x + y, StandardForm]

Out[3]=  <img src="/webMathematica/MSP?MSPStoreID=FileNameBase_874538807&
          MSPStoreType=image/gif" alt="Created by webMathematica" />
```

By default the images are GIF format; this can be changed by specifying the format as a third argument. Here the image is stored in JPEG format.

```
In[4]:= MSPFormat[ x + y, StandardForm, "JPEG"]

Out[4]=  <img src="/webMathematica/MSP?MSPStoreID=FileNameBase_1065528536&
          MSPStoreType=image/jpeg" alt="Created by webMathematica" />
```

It is also possible, though rather strange, to get a text-based format type (for example OutputForm) rendered into an image.

```
In[5]:= MSPFormat[ x + y, OutputForm, "GIF"]
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
Out[5]=  <img src="/webMathematica/MSP?MSPStoreID=FileNameBase_1061883558&
          MSPStoreType=image/gif" alt="Created by webMathematica" />
```

The width that is used for typeset images is set by the variable `$TypesetImageWidth`.

An alternative way to generate images is with the function `MSPExportImage`. For information on image generation see the section Displaying Mathematics and Graphics.

MathML

If the format is set to `MathMLForm`, the system will format the expression into MathML.

```
In[6]:= MSPFormat[ Sin[x]^2, MathMLForm]

Out[6]= <math>
        <mrow>
          <msup>
            <mi>sin</mi>
            <mn>2</mn>
          </msup>
          <mo>&af;</mo>
          <mrow>
            <mo>(</mo>
            <mi>x</mi>
            <mo>)</mo>
          </mrow>
        </mrow>
        </math>
```

In addition you can specify a content type of `RawMathML`. This can be useful to get the MathML for the `StandardForm` rendering of an expression. This output is raw in the sense that it does not use any reference to a plug-in, applet, or special browser that would be necessary to activate the MathML.

```
In[7]:= MSPFormat[ Sin[x]^2, StandardForm, "RawMathML"]

Out[7]= <math>
        <msup>
          <mrow>
            <mi>Sin</mi>
            <mo>&af;</mo>
            <mrow>
              <mo>[</mo>
              <mi>x</mi>
              <mo>]</mo>
            </mrow>
          </mrow>
          <mn>2</mn>
        </msup>
        </math>
```

More information on working with MathML is provided in the Applications: MathML section.

©1988-2006 Wolfram Research, Inc. All rights reserved.

9.1.5 MSPGetMessages

`MSPGetMessages[]`	return all messages generated by evaluations in the current kernel

`MSPGetMessages` gives a way to obtain any messages that have been generated by evaluations in the current kernel. It returns a list of strings, where each string contains the formatted contents of the message.

This function cannot be demonstrated in a normal evaluation; it must be part of a running server. The example Messages.jsp demonstrates the use of `MSPGetMessages`.

9.1.6 MSPGetPrintOutput

`MSPGetPrintOutput[]`	return the text of all print statements evaluated by the current kernel

`MSPGetPrintOutput` gives a way to obtain the output of all print statements that have been processed by the current kernel. It returns a list of strings, where each string contains the formatted contents of the message.

This function cannot be demonstrated in a normal evaluation; it must be part of a running server. The example Messages.jsp demonstrates the use of `MSPGetPrintOutput`.

9.1.7 MSPGetUploadFile

`MSPGetUploadFile[]`	process the HTTP headers to receive an uploaded file of up to 4 MB
`MSPGetUploadFile[ SizeLimit -> 5000]`	upload a file, with a limit of 5000 bytes on the upload

This provides a useful utility function for uploading files from the client to the server using multipart/form-data submissions. The contents of the file are saved in a new file on the server, and the name of this file is returned. The file on the server will eventually be cleared in a way similar to the clearing of image and other temporary files. The result of `MSPGetUploadFile` is a list of rules that show the filename on the server, the original filename on the client, and the content type.

The function cannot be demonstrated since it must really be part of an actual HTTP transaction with appropriate information sent from the client. This example just simulates the way that the function can be used with *Mathematica* programming. There are examples of using `MSPGetUploadFile` in the examples section.

```
In[1]:= Needs["MSP`"]

In[2]:= MSPGetUploadFile[]

Out[2]= {FileName → MSPStore2349287_0_1,
         OriginalFileName -> C:\last.dat, ContentType -> text/plain}
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

The filename can be extracted with the typical *Mathematica* commands used for working with rules.

```
In[4]:= "FileName" /. %

Out[4]= MSPStore2349287_0_1
```

Note that parameters that are present in the HTTP headers become available as `$$` variables after the use of `MSPGetUploadFile` or `MSPGetUploadFileList`. After this they can be used in the normal way.

There is a maximum size for a file that can be uploaded by `MSPGetUploadFile`; by default this is 4MB. If you wish to change this limit you can use the option `SizeLimit`.

Exceptions

`MSPGetUploadFile` can throw the following exception.

`MSPException["FileUploadError"]` in case of some error, for example, if there is not a single file to be uploaded or if the HTTP headers are malformed in some way.

9.1.8 MSPGetUploadFileList

`MSPGetUploadFileList[]`	process the HTTP headers to receive several uploaded files of up to 4 MB
`MSPGetUploadFileList[ SizeLimit -> 5000]`	upload files, with a limit of 5000 bytes on the upload

`MSPGetUploadFileList` is very similar to `MSPGetUploadFile`, except that it works for several files.

The function cannot be demonstrated since it must really be part of an actual HTTP transaction with appropriate information sent from the client. This example just simulates the way that the function can be used with *Mathematica* programming.

```
In[1]:= Needs["MSP`"]

In[2]:= MSPGetUploadFileList[]

Out[2]= {{FileName → MSPStore2349287_0_1, OriginalFileName -> C:\last1.dat,
          ContentType -> text/plain}, {FileName → MSPStore2349287_0_2,
          OriginalFileName -> C:\last2.dat, ContentType -> text/plain}}
```

Note that parameters that are present in the HTTP headers become available as `$$` variables after the use of `MSPGetUploadFile` or `MSPGetUploadFileList`. After this they can be used in the normal way.

There is a maximum size for a file that can be uploaded by `MSPGetUploadFileList`; by default this is 4MB. If you wish to change this limit, you can use the option `SizeLimit`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Exceptions

MSPGetUploadFileList can throw the following exception.

MSPException["FileUploadError"] in case of some error, for example, if there is not a single file to be uploaded or if the HTTP headers are malformed in some way.

9.1.9 MSPInclude

MSPInclude[*page*]	load the specified MSP script

MSPInclude allows one page to include the results of processing another page. Because it is not supported in the JSP version of web*Mathematica*, its use is discouraged. Similar functionality can be obtained by using jsp:include.

Exceptions

MSPInclude can throw the following exception.

MSPException["MSPInclude"] in case the function is called from within a JSP.

9.1.10 MSPLive3D

MSPLive3D[*graphics*]	include a LiveGraphics3D graphical applet in an HTML page

MSPLive3D is a convenient way to work with the LiveGraphics3D graphics applet. This applet displays *Mathematica* three-dimensional graphics and provides support for features such as interactive rotation and resizing. It is documented in the LiveGraphics3D section.

```
In[1]:= Needs["MSP`"]
```

The argument to MSPLive3D must be something that evaluates to a 3D graphics object, such as a Plot3D command.

```
In[2]:= MSPLive3D[ Graphics3D[ Line[ {{0, 0, 0}, {1, 1, 1}}]]]

Out[2]= <applet archive="/webMathematica/
          Resources/applets/live.jar" code="Live.class"
        width="400" height="400" align="top">
        <param name="input" value= "Graphics3D[Line[{{0., 0., 0.}, {1., 1., 1.}}], {}]
        "/>
        </applet>
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

9.1.11 MSPPageDirectory

MSPPageDirectory[]	returns the full path of the directory in which the current script is being processed

MSPPageDirectory returns the full path of the directory in which the current page is being processed.

```
In[1]:= Needs["MSP`"]

In[2]:= MSPPageDirectory[]

Out[2]= C:\Program Files\jakarta-tomcat\webapps\webMathematica\Examples
```

MSPPageDirectory can be used to load data files that live in the same directory as the current script. It should be noted that locating files in the script directory may mean that they are visible to an HTTP request. The use of MSPPageDirectory to load data files is discussed in the section File I/O.

9.1.12 MSPPageOptions

MSPPageOptions sets global options concerning the current page. The option ContentType sets the return content type, which is by default text/html. The option MinimumVersion sets the minimum version of the web*Mathematica* technology that can be used for this page.

ContentType

In this example the ContentType option is set to return MathML. If the browser is configured correctly, it will launch an appropriate MathML helper application.

```
<msp:evaluate>
    MSPPageOptions[ ContentType -> "text/mathml"]
</msp:evaluate>

<msp:evaluate>
    MSPFormat[ Integrate[ 1/(1-x^3),x], StandardForm, RawMathML]
</msp:evaluate>
```

The ContentType option provides similar functionality to MSPReturn. It is different in that it returns the entire page, whereas MSPReturn returns only its first argument. This topic is discussed in greater length in the Returning General Content section.

MinimumVersion

In this example the MinimumVersion option is set to require that the page should be run in web*Mathematica* Version 2.0 or higher. Otherwise an MSPException will be thrown.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<msp:evaluate>
    MSPPageOptions[ MinimumVersion -> 2.0]
</msp:evaluate>
```

This option is provided for use in future versions of web*Mathematica*.

9.1.13 MSPReturn

MSPReturn[*result, content*]	return the result with the specified MIME content type
MSPReturn[*result, content, filename*]	set the filename associated with the response

MSPReturn allows a page to return something that is not an HTML result. For example, a page can return a *Mathematica* notebook directly to the client.

```
<msp:evaluate>
    MSPReturn[ Notebook[ Cell[ "Hello", "Title"]],
     "application/mathematica"]
</msp:evaluate>
```

The three-argument form of MSPReturn is useful when you wish to set the filename associated with the response.

```
<msp:evaluate>
    MSPReturn[ Notebook[ Cell[ "Hello", "Title"]],
            "application/mathematica", "mynotebook.nb"]
</msp:evaluate>
```

In this case the client might try to use a filename of mynotebook.nb. It should also be noted that for some clients, such as Internet Explorer, setting the filename header can cause the display of two Open or Save dialog boxes. This topic is discussed in greater length in the Returning General Content section.

9.1.14 MSPRootDirectory

MSPRootDirectory[]	return the full path of the root directory of web*Mathematica*

MSPRootDirectory returns the full path of the root directory of the web*Mathematica* web application.

```
In[1]:= Needs["MSP`"]

In[2]:= Integrate[1];

In[2]:= MSPRootDirectory[]
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
Out[2]= C:\Program Files\jakarta-tomcat\webapps\webMathematica
```

9.1.15 MSPSetDefault

`MSPSetDefault[ var, value]`	set the variable *var* to *value* if it does not have a value

This is a utility function for setting the default values of variables.

```
In[1]:= Needs["MSP`"]
```

Here `$$var` has a value, so its value is not modified.

```
In[2]:= $$var = "5.6"; MSPSetDefault[ $$var, "foo"]; $$var

Out[2]= 5.6
```

If `$$var` has no value, `MSPSetDefault` will set it.

```
In[3]:= Clear[$$var]; MSPSetDefault[ $$var, "foo"]; $$var

Out[3]= foo
```

Exceptions

`MSPSetDefault` can throw the following exceptions.

`MSPException["ParseError"]` if the value cannot be interpreted by *Mathematica*.

`MSPException["SecurityError"]` if the value does not pass the security test.

`MSPException["ValueError"]` if the value is not a string. This indicates a programmatic error by the page author.

`MSPException["VariableError"]` if the variable is not a *Mathematica* symbol. This indicates a programmatic error by the page author.

9.1.16 MSPSessionVariable

`MSPSessionVariable[ sym]`	declare the variable *sym* to be a session variable, with the initial value `Null`
`MSPSessionVariable[ sym, value]`	set the initial value of the session variable *sym* to be *value*

This is a scoping construct for declaring a variable to be a session variable. The values of a session variable will be stored in a session managed by the servlet container. These sessions are a standard feature of modern web servers/browsers and are used to hold state on a server. This means that a value will live from one call of the server

©1988-2006 Wolfram Research, Inc. All rights reserved.

to another. The use of session scoping of variables was described previously and an example is shown in Session.jsp.

9.1.17 MSPShow

MSPShow[*graphics*]	include a graphical image in an HTML page

MSPShow is the main way to include graphical results from *Mathematica* for inclusion in an HTML page. A further discussion on formatting mathematics and graphics is given in the section on Displaying Mathematics and Graphics.

```
In[1]:= Needs["MSP`"]
```

The argument to MSPShow can be something that evaluates to a graphics object, such as a Plot command. Note that the DisplayFunction has been modified to hide actual graphical output at this stage.

```
In[2]:= MSPShow[ Plot[ Sin[x], {x, 0, 2}]]

Out[2]=  <img src="/webMathematica/MSP?MSPStoreID=FileNameBase_448181998&
          MSPStoreType=image/gif" alt="Created by webMathematica" />

In[3]:= MSPShow[ Graphics[ Line[ {{0, 0}, {1, 1}}]]]

Out[3]=  <img src="/webMathematica/MSP?MSPStoreID=FileNameBase_490423745&
          MSPStoreType=image/gif" alt="Created by webMathematica" />
```

MSPExportImage provides an alternative way to generate image output. More information on image generation is available in the section Displaying Mathematics and Graphics. An alternative way to render graphics that does not involve images is to use SVG.

9.1.18 MSPShowAnimation

MSPShowAnimation[{gra_1, gra_2,...}]	generate a GIF animation from a list of graphical objects

MSPShowAnimation is a convenient way to generate animated GIF images. The argument must be something that evaluates to a list of graphics objects. A further discussion on formatting mathematics and graphics is given in the section on Displaying Mathematics and Graphics.

```
In[1]:= Needs["MSP`"]

In[2]:= MSPShowAnimation[ Table[ Plot[Sin[x + i], {x, 0, 4 Pi}], {i, 0., 2 Pi - Pi / 4, Pi / 4}]]

Out[2]=  <img src="/webMathematica/MSP?MSPStoreID=FileNameBase_726239281&
          MSPStoreType=image/gif" alt="Created by webMathematica" />
```

Because generating animated GIFs can be a time-consuming process, it may be necessary to modify the KernelTimeLimit parameter.

©1988-2006 Wolfram Research, Inc. All rights reserved.

MSPShowAnimation uses the *Mathematica* Export command, which can be modified with a number of options. These are described in the online help for Export under Further Examples. The variable $ShowAnimationOptions holds the options for MSPShowAnimation. For example, the following causes MSPShowAnimation to generate animations that loop, spending two seconds on each frame.

```
In[3]:= $ShowAnimationOptions = {"Loop" → True, "AnimationDisplayTime" → 2}

Out[3]= {Loop → True, AnimationDisplayTime → 2}
```

SVG provides an alternative way to get animated graphical features.

9.1.19 MSPToExpression

MSPToExpression[*var*]	interprets the variable *var*
MSPToExpression[*var, fmt*]	uses the format *fmt* to interpret *var*
MSPToExpression[*var, fmt, head*]	returns the result wrapped in *head*

MSPToExpression is the web*Mathematica* version of ToExpression; it provides the same functionality for turning strings into *Mathematica* input. However, it provides some additional features. The most important feature is that it carries out a security check on its input. As described in the Security section, web*Mathematica* has a fully configurable security system, which is designed to prevent commands that may pose a security risk. Another difference is that MSPToExpression throws exceptions in cases where its argument cannot be interpreted or the argument is not a string.

You should always use MSPToExpression rather than ToExpression in your code because it provides additional security.

```
In[1]:= Needs["MSP`"]

In[2]:= SetSecurity[];
```

Here the variable $$var is assigned to the value "5+7"; note that the value is a string.

```
In[3]:= $$var = "5+7";
```

When MSPToExpression evaluates, the parsed value of $$var is returned.

```
In[4]:= MSPToExpression[ $$var]

Out[4]= 12
```

If a variable has no value, an MSPException is thrown.

```
In[5]:= Clear[ $$var]; MSPToExpression[ $$var]

        Throw::nocatch :
         Uncaught Throw[$$var, MSPException[NoValueError]] returned to top level.

Out[5]= Hold[Throw[$$var, MSPException[NoValueError]]]
```

Like ToExpression, MSPToExpression can take a format type for interpretation.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
In[6]:= $$var = "sin(x)"; MSPToExpression[ $$var, TraditionalForm]

Out[6]= Sin[x]
```

If the input value cannot be interpreted, an `MSPException` is thrown.

```
In[7]:= $$var = "Sin["; Catch[ MSPToExpression[ $$var], _MSPException, List]
        ToExpression::sntx : Syntax error in or before "Sin[".

Out[7]= {{$$var, Sin[}, MSPException[ParseError]}
```

If the input value does not pass the security test, an `MSPException` is thrown.

```
In[8]:= $$var = "ReadList[\"/etc/passwd\"]";
        Catch[ MSPToExpression[ $$var], _MSPException, List]

Out[8]= {{$$var, ReadList["/etc/passwd"]}, MSPException[SecurityError]}
```

Input can also be given in MathML.

```
In[9]:= $$e = "<math><msqrt><mfrac><mi>x</mi><mi>y</mi></mfrac></msqrt></math>";

In[10]:= MSPToExpression[ $$e]

Out[10]=  x
          ─
          y
```

`MSPToExpression` can be used on strings that are computed from input.

```
In[11]:= $$e = "a,b,c"; MSPToExpression["{" <> $$e <> "}"]

Out[11]= {a, b, c}
```

Exceptions

`MSPToExpression` can throw the following exceptions.

`MSPException["NoValueError"]` if the variable does not have a value. This indicates a programmatic error by the page author.

`MSPException["ParseError"]` if the value cannot be interpreted by *Mathematica*.

`MSPException["SecurityError"]` if the value does not pass the security test.

`MSPException["ValueError"]` if the value is not a string. This would indicate a programmatic error by the page author.

`MSPException["VariableError"]` if the variable is not a *Mathematica* symbol. This indicates a programmatic error by the page author.

©1988-2006 Wolfram Research, Inc. All rights reserved.

9.1.20 MSPURLStore

MSPURLStore[*data, type*]	store a string of data of the specified type and return a URL reference
MSPURLStore[*data, type, filename*]	store a string of data of the specified type and return a URL reference that contains a suggested filename

MSPURLStore stores a string of formatted data in the MSP servlet and returns a URL that can be used to retrieve the data. It is an extension of the mechanism for storing images. It can be used to hold input for some plug-in or applet in the client or it could be formatted into an img tag.

```
In[1]:= Needs["MSP`"]

In[2]:= m = DisplayString[Graphics[Line[ {{0, 0}, {1, 1}}]], "JPEG"];

In[3]:= MSPURLStore[m, "image/jpeg"]

Out[3]= /webMathematica/MSP?MSPStoreID=FileNameBase_97360396&MSPStoreType=image/jpeg
```

The result is a string that can be used as a URL referring to the MSP servlet, which can be used to retrieve the data from the store.

A third argument to MSPURLStore sets a filename in the URL. There are several uses for this functionality, such as choosing a helper application for the client. Note that no actual file with this name is created. The name is just placed in the URL.

```
In[4]:= MSPURLStore[m, "image/jpeg", "file.gif"]

Out[4]= /webMathematica/MSP/file.gif?MSPStoreID=
          FileNameBase_846174849&MSPStoreType=image/jpeg
```

This topic is discussed further in the Returning General Content section. You should remember that the result of MSPURLStore can be embedded in other formatting functions as described in the section MSP Functions Returning Images.

9.1.21 MSPValue

MSPValue[*var*]	returns the value of *var*; it returns a null string if *var* has no value
MSPValue[*var*, *def*]	returns the value of *var*; it returns *def* if *var* has no value

This is a utility function that is useful for extracting the value of variables.

```
In[1]:= Needs["MSP`"]
```

The value of the variable $$expr is returned.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
In[2]:= $$expr = 56; MSPValue[ $$expr]

Out[2]= 56
```

If `$$expr` has no value, then a null string is returned.

```
In[3]:= Clear[$$expr]; MSPValue[ $$expr]

Out[3]=
```

Here a default value is returned.

```
In[4]:= Clear[$$expr]; MSPValue[ $$expr, "x+y"]

Out[4]= x+y
```

Exceptions

`MSPValue` can throw the following exceptions.

`MSPException["ValueError"]` if the value is not a string. This would indicate a programmatic error by the page author.

`MSPException["VariableError"]` if the variable is not a *Mathematica* symbol. This would indicate a programmatic error by the page author.

9.1.22 MSPValueQ

`MSPValueQ[`var_1,var_2, ...`]`	returns `True` if all the variables have values

This is a utility function that tests whether variables have values.

```
In[1]:= Needs["MSP`"]
```

The variable `$$expr` has a value, so the result is `True`.

```
In[2]:= $$expr = 56; MSPValueQ[ $$expr]

Out[2]= True
```

Now `$$expr` has no value, so the result is `False`.

```
In[3]:= Clear[$$expr]; MSPValueQ[ $$expr]

Out[3]= False
```

All of the variables must have values for `MSPValueQ` to return `True`.

```
In[4]:= $$expr = 56; Clear[ $$var]; MSPValueQ[ $$expr, $$var]

Out[4]= False
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

9.1.23 $ExportImageOptions

$ExportImageOptions holds the options used for generating images when MSPExportImage is used. This uses the *Mathematica* Export command, and the possible options are described in the online help for Export under Further Examples. In this example the options are set to indicate that white is transparent; this gives the image a transparent background. The default value of $ExportImageOptions is {}.

```
<msp:evaluate>
    Block[{MSP`Utility`ImageQuality=2,
                $ExportImageOptions={"Transparency"→GrayLevel[1]}},
        MSPExportImage[
     Plot[Sin[x],{x,0,2Pi},Background->GrayLevel[1]]]
         ]
</msp:evaluate>
```

More information on image generation is available in the section Displaying Mathematics and Graphics.

9.1.24 $MSPFormatType

$MSPFormatType is the default format type used by MSPFormat. The default value is OutputForm.

9.1.25 $PageWidth

Expressions are formatted with the page width set by the variable $PageWidth. The default value is 78.

9.1.26 $PathInfo

$PathInfo is a *Mathematica* variable that holds the setting of the path information for the request.

9.1.27 $QueryString

$QueryString is a *Mathematica* variable that holds the setting of the query string for the request.

9.1.28 $ScriptName

$ScriptName is a *Mathematica* variable that holds the name of the top-level script. Note that this can also be given as a parameter on the URL.

©1988-2006 Wolfram Research, Inc. All rights reserved.

9.1.29 $ServletConfig

`$ServletConfig` holds a reference to the `ServletConfig` object of the servlet transaction. It is possible to make *J/Link* calls on this object as documented in the section Server APIs.

9.1.30 $ServletRequest

`$ServletRequest` holds a reference to the `HTTPServletRequest` object of the servlet transaction. It is possible to make *J/Link* calls on this object as documented in the section Server APIs.

9.1.31 $ServletResponse

`$ServletResponse` holds a reference to the `HTTPServletResponse` object of the servlet transaction. It is possible to make *J/Link* calls on this object as documented in the section Server APIs.

9.1.32 $ShowAnimationOptions

`$ShowAnimationOptions` keeps options used for generating animated GIFs with `MSPShowAnimation`. These are the options that are passed to the *Mathematica* `Export` command and are described in the online help for `Export` under Further Examples. In this example, the animation is set to loop with a time of two seconds between each frame. (It uses `Block` to restore the option setting after the computation has finished.) The default value of `$ShowAnimationOptions` is `{Loop -> True}`.

```
<msp:evaluate>
    Block[{$ShowAnimationOptions=
            {"Loop"→True,"AnimationDisplayTime"→2}},
        MSPShowAnimation[ Table[Plot[Sin[x+i],{x,0,2Pi}],
                {i,0,2Pi-Pi/$$n,Pi/10}]]
        ]
</msp:evaluate>
```

9.1.33 $TypesetImageWidth

`$TypesetImageWidth` sets the width in pixels for images of typeset expressions. The default value is `800`.

9.1.34 $WebApplication

`$WebApplication` gives the name of the web application in which web*Mathematica* has been installed. Normally this is "`/webMathematica`", but it could be different. Using this definition allows you to write portable code, which is easier to move from one web application to another.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
In[1]:= Needs["MSP`"];

In[2]:= $WebApplication

Out[2]= /webMathematica
```

One use of `$WebApplication` is to load resources such as images or applets from the root of the web*Mathematica* web application. `$WebApplication` is used to load applets in the TextApplet.jsp example.

9.2 MSP Taglib Reference

The six tags that are provided by the MSP taglib are discussed in this section. There are many examples of their usage, in particular those described in the section Basic Examples.

`msp:allocateKernel`	allocate a *Mathematica* kernel for computation
`msp:evaluate`	evaluate input to *Mathematica* and insert the result in the output page
`msp:get`	get the result of a *Mathematica* computation and use it to set a Java expression
`msp:set`	set a *Mathematica* variable with the value of a Java expression
`msp:includeClassicMSP`	include an MSP script
`msp:forwardClassicMSP`	forward to an MSP script

The most basic usage of the MSP taglib is for a JSP to wrap an `msp:allocateKernel` tag around one or more `msp:evaluate` tags.

```
...
<msp:allocateKernel>

    <msp:evaluate>
        ...
    </msp:evaluate>

    <msp:evaluate>
      ...
    </msp:evaluate>

</msp:allocateKernel>
...
```

The following is a description of the operation of these tags. It provides a summary of the steps in processing a JSP that uses the MSP taglib.

©1988-2006 Wolfram Research, Inc. All rights reserved.

9.2.1 msp:allocateKernel (tag open)

The msp:allocateKernel tag is used to obtain a *Mathematica* kernel for computation. The open tag takes the following steps for preprocessing the page.

Determine the Pool

msp:allocateKernel first determines what pool to use for the request based on the name of the JSP. If no pool was named, the General pool is used.

Allocate the Kernel

A *Mathematica* kernel is requested from the kernel pool. The pool maintains a collection of *Mathematica* kernels waiting for computations. If no kernel is available, the system waits until one is ready. Using a pool allows the system to share *Mathematica* kernels across multiple requests, which leads to a faster response time for the system.

Note that each request may get a completely different kernel. You cannot rely on saving anything in your *Mathematica* kernel and restoring it the next time.

Assign Input Variables

Any input variables that were sent with the request are then passed to the *Mathematica* kernel with their values. For a variable sym and value fun, a *Mathematica* assignment $$sym = "fun" is made. This ensures that the value is a *Mathematica* string, an inert object that will not evaluate without some special action. Note that input elements are not the only sources of variables. For example, an image map may cause transmission of variables. web*Mathematica* renames these input variables, and this helps to ensure that they do not interfere with your *Mathematica* code.

Each variable is scanned to verify that it is a valid *Mathematica* symbol. Any "." character is replaced by a backquote (`), and any underscore (_) is replaced with a "U". This mapping of names is consistent with the way that *J/Link* maps names.

Here are some samples of renamed variables.

Server Variable	*Mathematica Symbol*
input	$$input
name.x	$$name`x
var_x	$$varUx

Each variable is then validated to ensure it only contains letters or digits as well as the dollar ($) and backquote (`) characters. This prevents an attack that sends a variable starting with an exclamation (!) character. This would be potentially dangerous because it might cause *Mathematica* to launch an operating system shell.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Each value is turned into a *Mathematica* string. For this, any backslash (\) and doublequote (") characters are escaped with additional backslash (\) characters. If the value starts with an exclamation (!), a space is added. Finally, doublequotes (") are added around the result.

Server Value	*Mathematica Value*
`Sin[x+y]`	`" Sin[x+y] "`
`!myBoolean`	`" !myBoolean"`
`"\foo\bar "`	`" \" \\ foo \\ bar \" "`

Assign Parameters

Assignments to `$ServletRequest`, `$ServletResponse`, `$ServletConfig`, `$ScriptName`, `$Path Info`, and `$QueryString` appropriate for this request are made in the kernel.

Initialization

The settings of `$Context` and `$ContextPath` are saved, and the lists used to store messages and print output are initialized.

9.2.2 `msp:evaluate`

The `msp:evaluate` tag exists to evaluate *Mathematica* commands inside of a JSP. You can use the full range of MSP functions inside an `msp:evaluate` tag. Each tag uses the kernel that was allocated by the `msp:allocateKernel` tag to evaluate its contents. If no kernel has been allocated, the request is forwarded to an error page.

If any `MSPException` is thrown, it will be caught by the processing code, and some suitable error message will be inserted.

If any `MSPReturn` command is evaluated, processing of the current evaluation and all other evaluations is terminated immediately, and its argument is returned directly from the JSP. If no `MSPReturn` command is encountered, the result of the evaluation is inserted into the output stream.

The processing of each evaluation is wrapped in `TimeConstrained`, the time parameter being computed from the `KernelTimeLimit` configuration parameter.

If *Mathematica* generates any messages or print output, these are stored so they can be retrieved with MSPGetMessages and MSPGetPrintOutput respectively.

The result of the `msp:evaluate` tag will be formatted and returned in the result. In the example below, the current date will appear in the output page.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<msp:evaluate>
    Date[]
</msp:evaluate>
```

If you wish to calculate more than one result in an `msp:evaluate` tag the different steps must be separated with a semicolon ';'. The result of the last computation will be formatted and appear in the output page. In the example below, the numerical result of `x+y` will appear.

```
<msp:evaluate>
    x = Sin[5.6];
    y = Sqrt[x];
    x+y
</msp:evaluate>
```

More information on formatting of the result of `msp:evaluate` appears in the section on Evaluation Formatting.

It should be noted that within the scope of an `msp:allocateKernel` the same kernel will be used for all the tags `msp:evaluate`, `msp:set`, and `msp:get`, thus any definitions and commands made in one will be visible in others. These definitions are cleared out by `/msp:allocateKernel`.

9.2.3 `msp:allocateKernel` (tag close)

When the `msp:allocateKernel` tag is closed, the following steps are taken for postprocessing.

Java Exceptions

If any Java exceptions were thrown while processing the JSP, these are caught and the kernel is shut down and restarted. The exception is then rethrown and it may be returned with the HTTP request.

MSPReturn

If a `MSPReturn` was encountered during an evaluation, its argument is returned instead of the normal output of the JSP.

Set ContentType

The content type is set. It is specified by a setting of the `ContentType` option of `MSPPageOptions` or by `MSPReturn`. The default is `text/html`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Clean the Kernel

The kernel is cleaned so that it can be used again. This involves clearing the values of parameters that were sent with the request and removing all symbols in the default context. In addition `$Context` and `$ContextPath` are restored to their initial values, any Java object references are removed, and any open streams are closed.

Release the Kernel

The kernel is released to the pool so that it can be used again.

9.2.4 `msp:set`

The `msp:set` tag exists to use Java to set a *Mathematica* symbol. Each `msp:set` tag uses the kernel that was allocated by the `msp:allocateKernel` tag to evaluate its contents. If no kernel has been allocated, the request is forwarded to an error page.

The tag takes the following required attribute.

`name`	the name of the *Mathematica* variable to hold the result

In addition, one of the following attributes must be set.

`booleanValue`	Java boolean value
`byteValue`	Java byte value
`char Value`	Java char value
`intValue`	Java int value
`longValue`	Java long value
`floatValue`	Java float value
`doubleValue`	Java double value
`objectValue`	Java reference value

In the following example, the *Mathematica* variable `var` is set by the Java `int num`.

```
<msp:allocateKernel>

    <msp:set name="var" intValue="<%= num %>" />

</msp:allocateKernel>
```

An example of `msp:set` is given above.

©1988-2006 Wolfram Research, Inc. All rights reserved.

It should be noted that within the scope of an msp:allocateKernel the same kernel will be used for all the tags msp:evaluate, msp:set, and msp:get, thus any definitions and commands made in one will be visible in others. These definitions are cleared out by /msp:allocateKernel.

9.2.5 msp:get

The msp:get tag exists to get a value from *Mathematica* into Java. Each msp:get tag uses the kernel that was allocated by the msp:allocateKernel tag to evaluate its contents. If no kernel has been allocated, the request is forwarded to an error page.

The tag takes three required attributes, which are described below.

name	the name of the Java variable to hold the result
type	the type of the result; note that this must be a reference object
value	the *Mathematica* command to be evaluated

In the following example, the Java variable dValue with type Double is set to the result of the *Mathematica* function Random[].

```
<msp:allocateKernel>

    <msp:get name="dValue" type="Double" value="Random[]" />

</msp:allocateKernel>
```

An example of msp:get is given above.

It should be noted that within the scope of an msp:allocateKernel the same kernel will be used for all the tags msp:evaluate, msp:set, and msp:get, thus any definitions and commands made in one will be visible in others. These definitions are cleared out by /msp:allocateKernel.

9.2.6 msp:includeClassicMSP

The msp:includeClassicMSP tag exists to include an MSP script into a JSP. MSP scripts are an older way of working with web*Mathematica* described in the section Classic web*Mathematica* Technology.

The tag takes two attributes, which are described below.

name	the name of the MSP script to include; this attribute is required
arguments	any arguments to be passed to the MSP script; this attribute is optional

The following example includes the result of the MSP script Examples/PlotSingle.msp. The arguments fun=Sin[x]&x1=10 are passed to the MSP script.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<msp:includeClassicMSP name="Examples/PlotSingle"
    arguments="fun=Sin[x]&x1=10" />
```

An example of `msp:includeClassicMSP` is given above.

9.2.7 msp:forwardClassicMSP

The `msp:forwardClassicMSP` tag exists to forward to an MSP script from a JSP. MSP scripts are an older way of working with web*Mathematica,* described in the section Classic web*Mathematica* Technology.

The tag takes two attributes, which are described below.

`name`	the name of the MSP script to include; this attribute is required
`arguments`	any arguments to be passed to the MSP script; this attribute is optional

The following example will forward to the MSP script `Examples/PlotSingle.msp`. The arguments `fun=Sin[x]&x1=10` are passed to the MSP script.

```
<msp:forwardClassicMSP name="Examples/PlotSingle"
    arguments="fun=Sin[x]&x1=10" />
```

9.2.8 Overview

An overview of the steps in processing a page can be found in the introductory section on web*Mathematica* Technology.

9.3 Site Configuration

This section summarizes how to configure a web*Mathematica* site. Most configuration information is held in the file `MSP.conf`. The contents of this file are used to initialize the system and individual *Mathematica* kernels; this is described in the *Mathematica* Initialization section.

web*Mathematica* can support multiple kernel pools. A kernel pool is a group of *Mathematica* kernels that can be specially configured and is documented in the Advanced Topics: Multiple Kernel Pools section. If you do not configure any extra pools, then only one pool—known as the `General` pool—is used.

©1988-2006 Wolfram Research, Inc. All rights reserved.

9.3.1 MSP.conf

`MSP.conf` is the main configuration file. In addition, any extra kernel pool can have its own configuration file. Configuration for a pool `Tests` is in the file `Tests.conf`. The pool configuration file can override settings in `MSP.conf`. If you do not set a parameter in a pool configuration file, its value is inherited from `MSP.conf`. A few parameters modify the operation of the entire site and only have any meaning in `MSP.conf`, and any setting in a pool configuration file will be ignored.

Let us consider each of the parameters in greater detail.

CheckToExpression

This parameter controls whether the *Mathematica* command `ToExpression` should carry out a security check if the first argument is a string that was sent with the web*Mathematica* request. By default, the test is carried out, but a setting of `false` disables the check.

```
CheckToExpression=false
```

Typical sites should not modify this feature. This topic is discussed in greater detail in the Security section.

This parameter modifies the site and can only be set in `MSP.conf`.

CollectStreams

This parameter controls whether input and output streams that were opened in the *Mathematica* kernel are automatically closed. By default, they are closed. Setting `CollectStreams` to `false` prevents streams from being closed.

```
CollectStreams=false
```

Typical sites will not want to modify this feature. This topic is discussed in greater detail in the Security section.

This parameter can appear in `MSP.conf` and in a pool-specific configuration file.

FileCheckPeriod

This parameter controls how frequently temporary files created by web*Mathematica* are deleted. It sets the minimum lifetime in milliseconds for temporary files. The following example sets the time to 1000000 milliseconds.

```
FileCheckPeriod=1000000
```

©1988-2006 Wolfram Research, Inc. All rights reserved.

This parameter modifies the site and can only be set in MSP.conf.

FrontEndLaunchFlags

To make images or to use notebook API functions, web*Mathematica* uses the *Mathematica* front end. To launch the front end, you may need to set special configuration parameters with FrontEndLaunchFlags.

For example, to tell the front end to connect to a different X server for display, you could do the following.

```
FrontEndLaunchFlags= -mathlink -display :1 -nogui -geometry
    1000x500+10+10
```

This instructs the front end to use DISPLAY 1 with fixed geometry and in a server mode.

This parameter can appear in MSP.conf and in a pool-specific configuration file.

FrontEndLaunchName

To make images or to use notebook API functions, web*Mathematica* uses the *Mathematica* front end. In very special circumstances it might be useful to set the name of the front end binary to an explicit value. Typically, this is not necessary because a suitable value is chosen automatically.

For example, to set the location of the front end you could do the following.

```
FrontEndLaunchName=D:\\Program Files\\Wolfram
    Research\\Mathematica\\5.2\\Mathematica.exe
```

This instructs web*Mathematica* to use a front end from the D: drive. Typically, it is not necessary to use this parameter.

This parameter can appear in MSP.conf and in a pool-specific configuration file.

ImageAltText

This parameter sets the value of the alt attribute of generated img tags. By default, the value is "Created by webMathematica".

```
ImageAltText="This image was created by my server."
```

This parameter can appear in MSP.conf and in a pool-specific configuration file.

©1988-2006 Wolfram Research, Inc. All rights reserved.

JLinkNativeLibraryDirectory

Certain servlet containers will not load the *J/Link* native library from inside the web application layout. For these it is possible to place the native library in a central location and specify this location in `MSP.conf`. An example is shown below.

```
JLinkNativeLibraryDirectory=/usr/local/jlink
```

Note that the directory specified by this parameter should be the directory that contains the `SystemFiles` directory that is typically found in `webMathematica/WEB-INF/lib`. For Linux the layout should look something like the following.

```
jlink
    SystemFiles
        Libraries
            Linux
                libJLinkNativeLibrary.so
```

This parameter modifies the site and can only be set in `MSP.conf`.

KernelAcquireLimit

This parameter specifies the number of times that each kernel can be taken from the kernel pool before being shut down. Generally it is a good idea to shut down each kernel at a regular, but not necessarily frequent, interval. In this example each kernel is restarted after 500 hits.

```
KernelAcquireLimit=500
```

This parameter can appear in `MSP.conf` and in a pool-specific configuration file.

KernelConnectLimit

This parameter specifies the time that web*Mathematica* waits while trying to launch a kernel. If a kernel cannot be launched in this time, an error is reported. In this example, each kernel is restarted after 100 seconds.

```
KernelConnectLimit=100000
```

This parameter can appear in `MSP.conf` and in a pool-specific configuration file.

©1988-2006 Wolfram Research, Inc. All rights reserved.

KernelInitialization

This optional parameter gives extra evaluations for the *Mathematica* kernel that are processed at the end of kernel initialization. These could be used to load special packages or to set details of the MSP functions. In this example, the command `Needs["MyApplication`"]` evaluates every time a kernel is launched.

```
KernelInitialization=Needs[ "MyApplication`"]
```

If you set definitions for symbols that are in the default context with `KernelInitialization`, they are cleared and the symbols are removed by the postprocessing code for each request. This also applies to packages that are loaded by `KernelInitialization` that are not written in the *Mathematica* package format to make proper use of contexts. Any definitions that are made must use their own context for saving names. You can do this by prepending the name with a context (for example `TestNameSpace`Compute`) or by making appropriate use of `BeginPackage[]` and `EndPackage[]`.

Another point about the use of the `KernelInitialization` parameter is that certain packages may require the front end in order to be initialized correctly. You can load these packages into web*Mathematica* with `Developer`UseFrontEnd`; this is shown below.

```
KernelInitialization=Developer`UseFrontEnd[Needs[ "MyApplication`"]]
```

This parameter can appear in `MSP.conf` and in a pool-specific configuration file.

KernelNumber

The `KernelNumber` parameter gives the number of kernels that are launched when the system starts. You must make sure that you have sufficient licenses for this number of kernels. When there are insufficient licenses, some kernels will fail to launch. This will be noted in the log file.

```
KernelNumber=2
```

This parameter can appear in `MSP.conf` and in a pool-specific configuration file.

KernelPools

This parameter is used to specify the names of additional pools of *Mathematica* kernels, as documented in the Multiple Kernel Pools section. This example specifies two kernel pools, `Examples` and `Test`.

```
KernelPools=Examples,Test
```

This parameter modifies the site and can only be set in `MSP.conf`.

©1988-2006 Wolfram Research, Inc. All rights reserved.

KernelTimeLimit

This parameter gives the maximum number of milliseconds for processing a page. When this time is exceeded, the kernel is shut down and restarted. In this example, the kernel is restarted if processing a single page takes more than 60 seconds. The log system is notified when a kernel is shut down.

```
KernelTimeLimit=60000
```

This parameter can appear in `MSP.conf` and in a pool-specific configuration file.

MathLinkArguments

This is used as the argument to the *J/Link* method `createKernelLink`. If it is not set, a default appropriate for the platform on which *Mathematica* is running will be used. Here is a typical setting.

```
MathLinkArguments=-linkname 'math -mathlink' -linkmode launch
```

`MathLinkArguments` gives the command for launching *Mathematica*. If any special command-line arguments need to be given, they can be set here. For example, if you wish to launch *Mathematica* with a special password file, use a `-pwfile` command-line option. Additionally, it may be necessary to give a full path to the executable that launches *Mathematica*.

This parameter can appear in `MSP.conf` and in a pool-specific configuration file.

MSPDirectory

This parameter is only useful for MSP scripts, an older style of working with web*Mathematica*, described in the chapter Classic web*Mathematica* Technology. It describes the location of MSP scripts, and if it is not set then a default will be used. This directory is put on the *Mathematica* `$Path`, so it can contain packages and other applications that MSP scripts might load.

```
MSPDirectory=/usr/local/mathematica/AddOns/Applications/MSP/MSPScripts
```

This parameter can appear in `MSP.conf` and in a pool-specific configuration file.

This setting is not meaningful for the JSP implementation of web*Mathematica* since JSPs are located inside the web application.

PackagesDirectory

This optional parameter specifies a directory that is added to the *Mathematica* variable `$Path`. More information on loading and installing packages is available in the section Mathematica Packages and Applications.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
PackagesDirectory=/usr/local/files/Code
```

This parameter can appear in MSP.conf and in a pool-specific configuration file.

PoolPath

This parameter is used to map JSP scripts to kernel pools, as documented in the Multiple Kernel Pools section. This example specifies that JSPs in the ExampleDir directory will use the pool Examples and those in the TestDir directory will use the pool Test.

```
PoolPath.Examples=ExampleDir
PoolPath.Test=TestDir
```

This parameter modifies the site and can only be set in MSP.conf.

RequestLog

This optional parameter can be used to suppress the output that web*Mathematica* makes to the log system. By default, a line of output is generated for each request. This is suppressed when RequestLog is set to false.

```
RequestLog=false
```

This parameter modifies the site and can only be set in MSP.conf.

SecurityConfigurationFile

This parameter allows you to modify the security model by making your own definitions for MSP`Utility`AllowedContexts, MSP`Utility`DisallowedContexts, MSP`Utility`AllowedSymbols, and MSP`Utility`DisallowedSymbols. These definitions should be placed into a file in the /WEB-INF/conf directory and the name of the file set by the configuration parameter SecurityConfigurationFile. For example, if the configuration information is in a file called SecurityConfiguration.m, the following should be added.

```
SecurityConfigurationFile=SecurityConfiguration.m
```

More information on security is found in a previous section.

This parameter can appear in MSP.conf and in a pool-specific configuration file.

©1988-2006 Wolfram Research, Inc. All rights reserved.

StaticFileExtensions

This parameter gives the file extensions and content type for static files that the server may return. It is used so that MSP scripts can refer to image (and other files) with relative URLs. Further discussion is available in the Including Static Files section. A typical setting is as follows.

```
StaticFileExtensions=gif:image/gif,jpg:image/jpeg,jpeg:image/jpeg
```

This parameter modifies the site and can only be set in `MSP.conf`.

This setting is not meaningful for the JSP implementation of web*Mathematica* since the static content is processed by the servlet container.

VerboseLogs

The `VerboseLogs` parameter controls the level of logging that is recorded by the system. A setting of `false` saves only serious errors and one item for each request to the site. A setting of `true` records more information.

```
VerboseLogs=true
```

This parameter modifies the site and can only be set in `MSP.conf`.

WebApplicationName

When certain servlet containers load web applications they rename directories. This can cause problems for web*Mathematica* since it needs to know the name of the web application for generating URLs. If the name is incorrect these URLs will be incorrect and problems, such as graphics not working, will result. This can be solved by setting the name of the web application in `MSP.conf`, as shown below.

```
WebApplicationName=webMathematica
```

This parameter modifies the site and can only be set in `MSP.conf`.

9.3.2 Security Configuration

The security system for web*Mathematica* can be customized. This is described in the Security section.

©1988-2006 Wolfram Research, Inc. All rights reserved.

9.3.3 X Server Configuration

Special configuration is often required to allow the front end to connect an X server. This is described in the Installation section under Configuring for the X Window System (Unix only). This is only an issue for running web*Mathematica* under Unix.

9.4 LiveGraphics3D

One of the useful features of web*Mathematica* is its integration with the `LiveGraphics3D` applet. This applet displays *Mathematica* three-dimensional graphics and provides support for features such as interactive rotation and resizing. It is shipped with web*Mathematica* and is used by the command `MSPLive3D`. The applet has been carefully developed so that it works in a wide range of different Java-enabled browsers.

The interface is given below.

user action	*applet reaction*
drag and press the left mouse button	rotate about an axis in the picture
release the left mouse button while dragging	spin about an axis in the picture
press the SHIFT key and drag vertically	zoom
press the SHIFT key and drag horizontally	rotate about an axis perpendicular to the picture
press the CONTROL key and drag vertically	change the focal length
press the CONTROL key and drag horizontally	change the strength of the stereo effect
press the META (ALT) key (or the right mouse button) and drag vertically	strip parts of the graphics
press the o key	write parameter settings in the Java console
press the s key	toggle between single and stereo views
press the HOME key	restore original perspective (no spinning)

In addition, the `MSPLive3D` command can set the `Magnification` parameter for the applet as follows.

```
<msp:evaluate>
    MSPLive3D[ Plot3D[ Sin[x y],{x,0,3},{y,0,3}], Magnification ->
     0.4]
</msp:evaluate>
```

More information on the `LiveGraphics3D` applet is available from the web site maintained by its author, Martin Kraus, at http://wwwvis.informatik.uni-stuttgart.de/~kraus/LiveGraphics3D.

©1988-2006 Wolfram Research, Inc. All rights reserved.

9.5 Dynamic HTML

When the web was first developed, it supported only distribution of static pages. The technology was extended to allow interactive access for dynamic content generation.

Fundamentally, the web is driven by its main protocol, HTTP (Hyper Text Transfer Protocol), which imposes certain constraints. Under HTTP, a client sends a request to a server that replies with a response. A crucial feature of HTTP is that it is stateless; that is, after processing a request, no record of that request is kept. Of course, state information can be maintained via some other mechanism; for example, the servlet API has methods for keeping state that can be used with `MSPSessionVariable`.

This reference section reviews server and client technologies for dynamic web content. Because this field changes very rapidly the survey is not intended to be exhaustive.

9.5.1 Server Technology

There are several server-side technologies for dynamic content. These include CGI scripting, Active Server Pages, server plug-ins, Perl scripting, and Java servlets and JavaServer Pages.

CGI Scripting

CGI scripts provided the original server technology for dynamic content. Under CGI, an executable, such as a shell script or compiled binary, is launched on every request.

This mechanism is limited in a number of key ways. It is relatively expensive since it requires launching a new CGI process for every request, which can cause scalability problems. One solution is to make the actual CGI script a lightweight process that communicates with its own server; many web solutions actually do this. Of course this requires nontrivial development and can result in something that is more complicated to use than other dynamic solutions.

Active Server Pages

Active Server Pages (ASPs) are a scripting language for dynamic web content, developed by Microsoft. They are quite common and powerful. At present, ASPs are not supported, but this will be continually reviewed.

Server Plug-ins

Most HTTP servers provide some type of extensibility that can be used to support special features for interactivity. The problem with this approach is that it is not very portable.

Perl Scripting

Perl is often used as a scripting language either with CGI or a server plug-in. Technologies exist to link *Mathematica* to Perl, but these are not as developed as is the technology for linking to Java.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Java Servlets and JavaServer Pages

Java Servlet technology provides a high-level API (programming interface) for working with HTTP requests. There are many ways that web servers can be enhanced to add a servlet engine. Solutions exist for all web servers and run on all major platforms.

The Servlet API is a high-level interface that provides functions both for maintaining information while the server is running and for working with HTTP requests and responses.

JavaServer Pages (JSPs) are a closely related technology that make it very convenient for servlets to return HTML; the server converts a JSP into a servlet, which is then executed.

web*Mathematica* is implemented with a mixture of Java Servlet and JSP technology.

9.5.2 Client Technology

Ultimately, any content is downloaded to a client where it is rendered. In a sense, the purpose of server technology is to prepare input for a client. In the client there are also various dynamic content technologies. These include HTML, JavaScript, and applets.

HTML

The fundamental content delivered by web servers is HTML, a tree-structured language formed from tags. At present, HTML is being transitioned into a stricter language, XHMTL, an XML application.

HTML is fundamental to the topic of dynamic web content, so here is a short primer on dynamic web content with HTML.

First, start with a basic HTML document.

```
<html>
<title>My Page</title>
<body>
<h1>My Page</h1>
<p>Welcome to my page.</p>
</body>
</html>
```

This could be downloaded from a web server and rendered in a web browser.

Active elements are added to HTML by `form` and `input` elements, which can be included inside an HTML document. Here is a `form` element.

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
<form action="http://myhost/active"  method="post">

</form>
```

The form has two attributes, an action attribute and a method attribute. When the form is activated, it will make a connection to this URL and use the post method.

Often the URL will be located on the same server from which the page was downloaded. In this case it is common to use a relative URL.

```
<form action="active"  method="post">

</form>
```

A form element may contain input elements, which add various buttons and input fields. Here is an example of a form with two input tags: the first allows us to enter some text, and the second causes the form to be submitted.

```
<form action="active"  method="post">

<input type="text" name="ARG1">

<input type="SUBMIT" name="button" value="Compute">

</form>
```

When the form is activated by the submit input tag, the browser makes a request to the URL referred to by the action attribute. It sends the name and value pairs from all of the input tags in the form. This is the most basic way to activate HTML.

One thing to remember about form elements is that the name/value pairs can be specified in the URL. You may have seen them in something like http://myserver/document?ARG1=10&ARG2=20.

JavaScript

JavaScript is a compact object-based scripting language for developing client and server internet applications. JavaScript code can be embedded directly in an HTML page. It can, for example, embellish the operation of form and input elements. One problem with JavaScript is that it is not uniform across different browsers. JavaScript can manipulate the browser and the documents that the browser holds. It can also interact with applets and plug-ins.

Some of the examples included with web*Mathematica* work with JavaScript.

©1988-2006 Wolfram Research, Inc. All rights reserved.

Applets

Applets are programs written in Java that can run in a Java-enabled browser. They are less tightly integrated with HTML than JavaScript but are probably easier to develop and can call on much of Java technology. As with JavaScript, some (especially older) browsers give incomplete and poor support for applets. Applets can call on the large collection of functions that are available in the Java programming language and can actually draw into the browser.

Some of the examples are designed to work with applets.

Future Developments

The major browsers, Internet Explorer, Mozilla, and Netscape Navigator, continue to develop new interactive technologies at a rapid pace. It may be advantageous for web*Mathematica* users to consider new technologies as they become available.

9.6 MathML Technology

This section gives a summary of the various technologies for working with MathML. For an up-to-date list of the technologies that work with MathML, visit http://www.w3.org/Math. Information about how to use MathML in web*Mathematica* is given in the MathML section. Following are brief descriptions of some of the key MathML technologies to integrate with web*Mathematica*.

9.6.1 MathPlayer

http://www.dessci.com/webmath/mathplayer/

MathPlayer is a MathML display engine for Internet Explorer on Windows, Version 5.5 and later. It is supported by the MathML stylesheet described above. Design Science produces a free-of-charge version of MathPlayer, which is available from their web site. MathPlayer supports the copying of MathML to the Clipboard, which can be pasted into some other application, such as *Mathematica*. This provides a simple but effective way to reuse MathML that you view over the web.

9.6.2 WebEQ

http://www.dessci.com/webmath/webeq/features.stm

WebEQ is a suite of tools for building web pages that involve dynamic math. Of particular interest to web*Mathematica* is WebEQ Input Control, a Java applet that can function as an easy-to-use graphical equation editor in a web browser. It provides a customizable toolbar that allows for easy entry of special characters and templates for entire structures.

One important use of WebEQ is to provide a template-based entry system for mathematical notation on the web. Since it can be specialized to particular subject matter, it is quite flexible and powerful.

©1988-2006 Wolfram Research, Inc. All rights reserved.

9.6.3 Mozilla

http://www.mozilla.org

Mozilla is an open-source web browser that is supported on a wide variety of platforms, including Windows, Macintosh, Linux, and Unix. The latest versions of Mozilla support MathML natively, and this is used by the MathML stylesheet as described above. Mozilla provides many of the traditional web browser features and is somewhat related to Netscape Navigator.

9.6.4 Amaya

http://www.w3.org/Amaya/

Amaya is a browser/authoring tool that allows you to publish documents on the web. It is available for Windows, Linux, and Solaris. Amaya supports MathML natively and this is used by the MathML stylesheet as described above. Amaya also provides native support for SVG, an XML application for drawing.

9.7 File Layout

The layout of various web*Mathematica* archives is given in this section.

web*Mathematica* Web Application

This is the file layout of the web*Mathematica* web application.

webMathematica	
index.html	
BrowseExamples	*webMathematica examples accessed via a design template*
Examples	
Computation	*Calculating examples*
Data	*Data loading examples*
DesignTemplates	*Examples with better design*
HTML	*HTML formatting examples*
MathML	*MathML examples*
PDF	*PDF generation examples*
SVG	*SVG examples*
XML	*XML examples*
Resources	
applets	
CSS	*HTML stylesheets*
Documentation	
ErrorPages	*Pages to display error text*
images	
banners	
buttons	
Template	
XSL	*XSLT stylesheets for MathML rendering*

©1988-2006 Wolfram Research, Inc. All rights reserved.

```
WEB-INF
    web.xml
    Applications
        Data                Sample data files
        ExampleUtilities    Basic example programs
        JLink               JLink Mathematica application
        MSP                 MSP Mathematica application
    conf
        mathpass            Sample webMathematica license file
        MSP.conf            Configuration file for webMathematica
        webMathematica.tld
    classes                 Location for additional classes
    lib                     webMathematica executable archives
        kernelpool.jar
        JLink.jar
        servlets.jar
        tags.jar
        SystemFiles         JLink native libraries
    MSPScripts              Classic webMathematica examples
    src                     webMathematica sources
```

Many of these files are provided as examples, documentation, and sources. In addition, there are files specific to all the platforms on which web*Mathematica* runs. If you wish to run in a more minimal environment it is possible to strip out many of these files. This is described in the section on Minimal Installation.

MSP *Mathematica* Application

The layout of files in the MSP application is as follows.

```
MSP
    MSP.m
    HTML.m
    SVG.m
    Security.m
    Kernel
        init.m
    Documentation
        English
            webMathematicaUserGuide.nb
```

9.8 Links

The links listed here were valid at the time this documentation was written.

9.8.1 *Mathematica* Technology

Main Wolfram Research site:
http://www.wolfram.com

©1988-2006 Wolfram Research, Inc. All rights reserved.

web*Mathematica*:
http://www.wolfram.com/products/webmathematica

web*Mathematica* release notes:
http://documents.wolfram.com/webmathematica/releasenotes

web*Mathematica* documentation updates:
http://documents.wolfram.com/webmathematica/updates

web*Mathematica* support:
http://support.wolfram.com/webmathematica

***Mathematica*:**
http://www.wolfram.com/products/mathematica

***Mathematica* products:**
http://www.wolfram.com/products

***J/Link*:**
http://www.wolfram.com/solutions/mathlink/jlink

***DatabaseLink*:**
http://documents.wolfram.com/mathematica/Add-onsLinks/DatabaseLink/

The Web Services Package:
http://documents.wolfram.com/mathematica/Add-onsLinks/WebServicesPackage/

LiveGraphics3D:
http://wwwvis.informatik.uni-stuttgart.de/~kraus/LiveGraphics3D.

9.8.2 *Mathematica* Packages

Writing packages:
http://documents.wolfram.com/v4/AddOns/StandardPackages/6.18.5.html

Writing and saving packages:
http://support.wolfram.com/mathematica/packages/writing/make.html

9.8.3 Java

Sun Java information:
http://java.sun.com

Sun JDK download for Windows/Intel, Linux/Intel, and Solaris:
http://java.sun.com/j2se/downloads/index.html

Mac OS X Java information:
http://www.apple.com/java/

©1988-2006 Wolfram Research, Inc. All rights reserved.

PA-RISC/HP-UX Java information and JDK download:
http://www.hp.com/java/

IBM-RISC/AIX Java information and JDK download:
http://www.ibm.com/developerworks/java/jdk/aix/

Alpha/Tru64 Java information and JDK download:
http://www.hp.com/java/

Itanium/Linux JDK developer release:
http://java.sun.com/j2se/downloads/index.html

9.8.4 Tomcat

Links to current versions:
http://documents.wolfram.com/webmathematica/updates/install/tomcat/current

Apache Tomcat download section:
http://jakarta.apache.org/site/binindex.cgi

Main Apache site:
http://jakarta.apache.org

9.8.5 Servers JSPs and Servlets

Main site for servlet technology:
http://java.sun.com/products/servlet/

Uses of servlets:
http://java.sun.com/products/servlet/industry.html

Main site for JSP technology:
http://java.sun.com/products/jsp/

Apache, HTTP server:
http://httpd.apache.org

JRun, servlet engine:
http://www.macromedia.com/software/jrun/

9.8.6 XML, HTML, MathML, and SVG

XML:
http://www.w3.org/XML/

HTML 4.0:
http://www.w3.org/TR/REC-html40/

©1988-2006 Wolfram Research, Inc. All rights reserved.

XHTML:
http://www.w3.org/TR/xhtml1/

MathML:
http://www.w3.org/Math/

MathML characters:
http://www.w3.org/TR/MathML2/chapter6.html

Unicode characters:
http://www.unicode.org/

MathPlayer:
http://www.dessci.com/webmath/mathplayer/

WebEQ:
http://www.dessci.com/webmath/webeq/features.stm

Mozilla:
http://www.mozilla.org

Amaya:
http://www.w3.org/Amaya/

SVG:
http://www.w3.org/Graphics/SVG/

Adobe SVG:
http://www.adobe.com/svg/

9.8.7 PDF Tools

PStill:
http://www.pstill.com

ps2pdf:
http://www.cs.wisc.edu/~ghost/doc/AFPL/8.00/Ps2pdf.htm

9.8.8 The X Window System

RealVNC:
http://www.realvnc.com/

TightVNC:
http://www.tightvnc.com/

Xvfb:
http://www.xfree86.org/4.3.0/Xvfb.1.html

©1988-2006 Wolfram Research, Inc. All rights reserved.